



**A SECURITY ARCHITECTURE
FOR
NET-CENTRIC ENTERPRISE
SERVICES (NCES)**

Version 0.3 (Pilot)

Defense Information Systems Agency (DISA)

March 1, 2004

Document Change Record

Version	Date	Description
NCES v0.3	03-Mar-2004	First public release

Table of Contents

1. EXECUTIVE SUMMARY	1
2. NOTATIONS AND TERMINOLOGY	2
2.1 <i>Notations</i>	2
2.2 <i>Terminology</i>	2
3. BACKGROUND	4
3.1 <i>Service Oriented Architectures</i>	4
3.2 <i>Net-Centric Enterprise Services (NCES)</i>	6
3.3 <i>Overview of XML Security Standards</i>	7
4. ARCHITECTURE OVERVIEW	10
4.1 <i>Security Challenges under SOAs</i>	10
4.2 <i>Summary of Architectural Requirements</i>	12
4.3 <i>Scope, Assumptions, and Limitations</i>	14
4.4 <i>Conceptual Enterprise Security Architecture</i>	15
4.5 <i>The Security CESS</i>	17
5. THE TRUST MODEL	23
5.1 <i>The Micro View</i>	23
5.2 <i>The Macro View</i>	25
6. SECURITY ARCHITECTURE WITHIN A TRUST DOMAIN	28
6.1 <i>Securing the Invocation Path</i>	28
6.2 <i>Service Chaining</i>	32
7. AUTHENTICATION	35
7.1 <i>Asserting the Authentication of End Users</i>	35
7.2 <i>SOAP Message Authentication</i>	37
8. AUTHORIZATION	42
8.1 <i>Authorization Architecture</i>	42
8.2 <i>Two Approaches for Making Policy Decisions</i>	44
8.3 <i>Policy Decision Implementation Considerations</i>	46
9. AUTHORIZATION POLICIES	48
9.1 <i>The RBAC Model</i>	48
9.2 <i>Looking Ahead: the Attribute Based Approach</i>	51
10. OTHER TOPICS	53
10.1 <i>Message Confidentiality</i>	53
10.2 <i>Use of DoD PKI</i>	53
10.3 <i>Beyond Trust Domain Boundaries</i>	56
11. FUTURE WORK	58

A. MESSAGE EXAMPLES.....	61
<i>A.1 SAML Assertion Element Created by Portal</i>	<i>61</i>
<i>A.2 Signed SOAP Request.....</i>	<i>61</i>
<i>A.3 SAML-P Authorization Decision Query.....</i>	<i>63</i>
<i>A.4 SAML-P Authorization Decision Response</i>	<i>64</i>
<i>A.5 XACML Policy Set.....</i>	<i>64</i>
B. REFERENCES	69

List of Figures

FIGURE 1 - SERVICE ORIENTED ARCHITECTURE	4
FIGURE 2 - UNDERLYING SECURITY TECHNOLOGY STACK	8
FIGURE 3 - CONCEPTUAL ENTERPRISE SECURITY ARCHITECTURE	16
FIGURE 4 – THE MICRO VIEW: WITHIN A SINGLE TRUST DOMAIN.....	24
FIGURE 5 – THE MACRO VIEW: ACROSS TRUST DOMAIN BOUNDARIES	26
FIGURE 6 - LOGICAL SECURITY ARCHITECTURE, SINGLE TRUST DOMAIN.....	28
FIGURE 7 - CHAINING OF INBOUND MESSAGE HANDLERS.....	30
FIGURE 8 - SERVICE CHAINING.....	33
FIGURE 9 - MESSAGING TERMS	35
FIGURE 10 - MESSAGE AUTHENTICATION	38
FIGURE 11 - SIGNED SOAP MESSAGE	39
FIGURE 12 - AUTHORIZATION ARCHITECTURE	43
FIGURE 13 - BASIC RBAC MODEL	49
FIGURE 14 - RBAC BASED POLICIES.....	51

List of Tables

TABLE 1 - CURRENTLY SUPPORTED STANDARDS AND THEIR VERSIONS.....	9
TABLE 2 - SECURITY CES TAXONOMY	18

1. EXECUTIVE SUMMARY

The emergence of Web Service technologies has triggered a major paradigm shift in distributed computing: from Distributed Object Architectures (DOAs) to Service Oriented Architectures (SOAs). Within the Department of Defense (DoD) Enterprise there has been a growing need for increased integration and collaboration among “Communities of Interest” (COIs), often across organizational boundaries. The DoD transformation towards Net-Centricity highlights the need even further. A common set of Core Enterprise Services (CESs) represent crucial infrastructure components that support this vision. SOAs are well positioned to become the key technology enabler for Net-Centricity due to their decentralized, loosely coupled, and highly interoperable architecture. Securing a SOA, however, faces new challenges that cannot be fully addressed by existing Information Assurance solutions. This document describes the drivers, challenges, and requirements for securing a SOA in the Net-Centric environment, and proposes a security architecture that meets the unique burden of securing a decentralized system.

This document presents the high-level reference architecture, and defines an abstract “Security CES” layer that encapsulates enterprise security functionality such as authorization and credential management. To help secure Net-Centric interactions among enterprise service consumers and providers, the Security CESs themselves are defined as Web Services that are standards-based, platform-independent, and technology-neutral. The document also introduces the concepts of an indirect or “brokered” trust model, and argues that such a trust model is necessary to support an environment involving decentralized, heterogeneous security infrastructure and policies.

The architecture is then described in detail, along with guidance for how such an architecture can be constructed using emerging industry standards such as WS-Security, SAML, XACML, and XML Digital Signatures. More importantly, this document defines additional processing rules that “profile” these standards for use in a DoD Enterprise environment. These processing rules allow the security architecture to achieve interoperability while leveraging an underlying foundation of DoD Enterprise security infrastructure such as Identity Management and PKI.

Just like the Web Service technologies it leverages, the security architecture presented in this document is still in its infancy. Some potential future work items are listed at the end of the document, and it is expected that the scope of this document will grow over time.

2. NOTATIONS AND TERMINOLOGY

2.1 Notations

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in IETF RFC 2119 [RFC 2119]. E.g.:

... they MUST only be used where it is actually required for interoperation or to limit behavior which has potential for causing harm (e.g., limiting retransmissions) ...

These keywords are thus capitalized when used to unambiguously specify requirements over protocol and application features and behavior that affect the interoperability and security of implementations. When these words are not capitalized, they are meant in their natural-language sense.

Fixed width texts used for file names, constants, <XML elements>, and code examples.

Example code listings appear like this.

Italics texts are used for variables and other type of entities that can change. Italics are sometimes also used for emphasized text or annotations.

Terms in ***italic bold face*** are intended to have the meaning defined in the glossary.

Underlined texts are used for URLs.

2.2 Terminology

The following terms are frequently used in this document and are briefly explained below using commonly accepted definitions in the security literature. The reference section contains a number of security related glossaries that are much more comprehensive [RFC2828] [WS-GLOS].

Identity: A set of attributes that uniquely identifies a system *entity* such as a person, an organization, a service, or a device.

Comment: Note that identities are not just for human users. Resources such as data providers and service providers may also have their own identities. Also note that an entity may have multiple identities (e.g. local, legal, organizational).

Identifier: A sufficiently unambiguous reference to an *identity* of a system entity.

Comment: Note the difference between an identity and an identifier.

Credential: A.k.a. *Authentication Credential*, data that is required to demonstrate the possession of something in order to establish a claim.

Comment: Credentials may be symmetric information (e.g. password, shared secret key, or biometrics) or asymmetric information (e.g. private and public keys). An identity may have more than one set of credentials. For example, a user may have a login/password, a digital certificate, and his/her biometrics as credentials.

Principal: A Principal is an entity that has a **network identity** (see below), that is capable of making decisions, and to which authenticated actions are done on its behalf. A principal may refer to human entities such as an individual user, an organization, or a legal entity; depending on the context it may also refer to non-human system entities such as a Web Service provider.

*Comment: This document makes a distinction between Principals and Identities. A principal may have multiple local identities in different **Administrative Domains**. For example, a user principal can have a work account called "JDoe" in his employer's network, and also a personal account called "John_Doe" issued by his Internet Service Provider (ISP).*

*Please note that this document deliberately chooses not to use the term **Subject** due to its overloaded meanings in different contexts.*

Network Identity: The abstraction of the global set of attributes composed from a Principal's existing accounts.

Comment: "Network Identity" and "Principal" are used interchangeably in this document, in that both of them denote an abstract "global identity" that consists of/ maps to a "local identity" in a specific security domain.

Trust Domain: This document defines a Trust Domain (TD) as a purely logical construct within which a single set of access control policies hold.

*Comment: This document deliberately chooses not to use the term **Security Domain** to refer to a **Trust Domain**, due to its overloaded meanings and the often confusion with **Administrative Domains**. A Trust Domain has nothing to do with administrative or network security boundaries. In fact, multiple TDs may reside in a single administrative domain.*

Service Provider: A system entity that serves as a logical "container" of one or more Web Service applications. A service provider may host one or more Web Services, and may consist of one or many physical machines. If necessary, a service provider may be assigned its own network identity and thus be considered a principal.

Service Consumer: A system entity that issues service requests and consumes returned information. Within a SOA, a service consumer is usually an application.

Comment: A service provider may be a consumer of other service providers.

3. BACKGROUND

3.1 Service Oriented Architectures

The emergence of Web Service* (WS) technologies has triggered a major paradigm shift in distributed computing. Architectures are quickly moving from DOAs using technologies such as CORBA, DCOM, DCE, and Java RMI, to SOAs using technologies such as SOAP, HTTP and XML. Under a SOA, a set of network-accessible operations and associated resources are abstracted as a “service”. The service is described in a standard fashion, published to a service registry, discovered by a service consumer, and invoked by a service consumer. Figure 1 illustrates the three steps of Publish, Discover and Invoke.

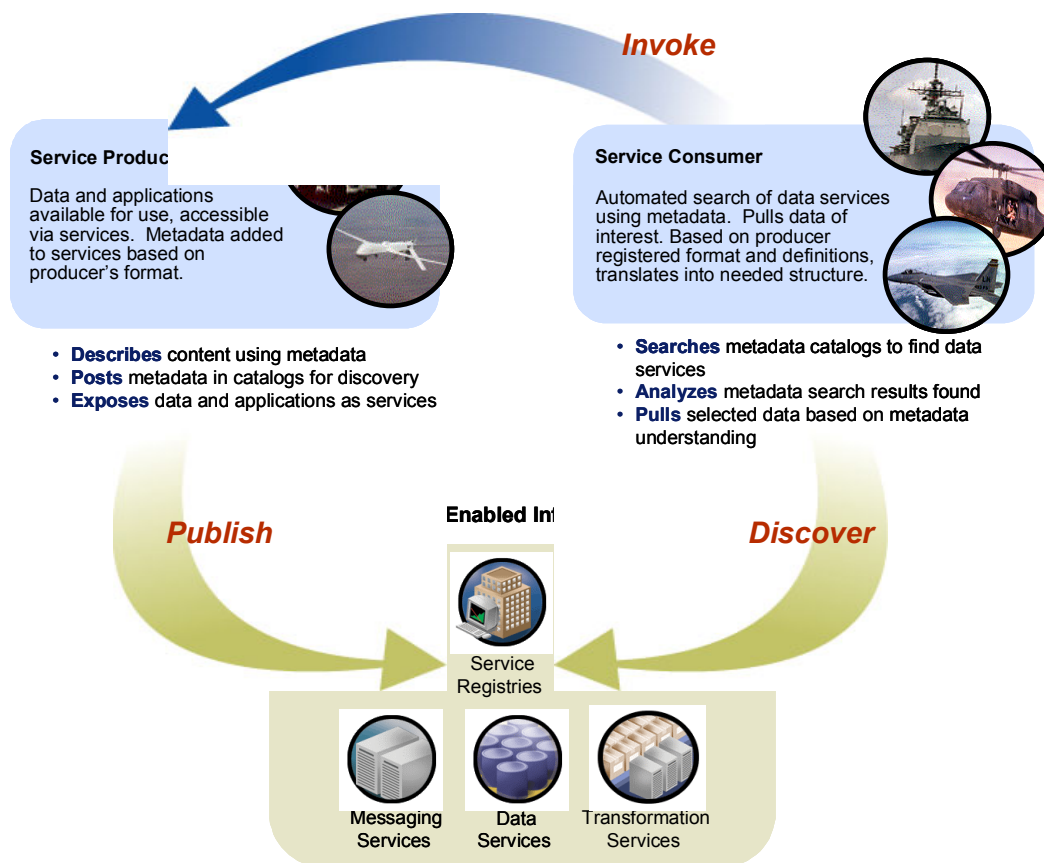


Figure 1 - Service Oriented Architecture

Three basic standards serve as the foundation of the Web Services protocol “stack”:

- **Simple Object Access Protocol (SOAP)** [SOAP] performs the low-level XML communications necessary for transmitting Web Service calls across the network.

*This architecture document refers to Web Services (with capital letters) as services using industry-accepted Web Service technology standards such as SOAP, WSDL, and UDDI, as opposed to general services offered over the Web.

SOAP provides a means of XML-based messaging between a service provider and a service consumer.

- **Web Service Definition Language (WSDL)** [WSDL] is an XML-based language that defines the functional interfaces for a Web Service. In other words, a WSDL document represents the official “contract” between service providers and their consumers. These WSDL interfaces are described first in abstract message structures, and then bound to a concrete transport protocol and a communication “endpoint”.
- **Universal Discovery, Description, and Integration (UDDI)** [UDDI] is an emerging standard for organizing and accessing a service registry (see Figure 1). A service registry serves as the yellow pages of a collection of Web Services, providing mechanisms for a service provider to publish its capabilities and for a service user to discover matching services.

A SOA offers several distinct advantages over traditional distributed computing technologies:

- **Maximum Interoperability** – The W3C and OASIS, among others, are currently defining Web Service standards that are entirely based on XML. This ensures that the standards are programming language-, platform-, and programming model-neutral. For example, a .NET Web Service client written in the procedural model of Visual Basic can readily invoke an Object-Oriented Web Service hosted by a Java 2 Enterprise Edition (J2EE) server on a Linux machine.
- **Loose Coupling** – Web Service standards define the functional interfaces that represent the minimal understanding between service consumer and service provider. Knowledge of the service provider is discovered dynamically from a service registry rather than statically coded in the client program.
- **Ubiquity** – Web Service calls are essentially XML messages sent over well-understood Internet protocols such as HTTP. These protocols represent the “least common denominator” of network protocol stacks and makes it easier to overcome firewall and infrastructure constraints. Web Services are likely to be the most viable option for inter-agency information sharing among different autonomous networks.

The migration toward more agile SOAs is not merely a technology push; there are also a number of key business drivers at work. In e-Business and e-Government alike, there is a growing need for increased integration and collaboration across organizational boundaries. Here are some domain scenarios:

- **E-Business / E-Gov Integration** – As businesses strive to keep costs down and become more agile in meeting customer demands, it is necessary to have a technology infrastructure that can enable “deep” integration in the supply chain. Within this scenario, complex systems such as Customer Relationship Management (CRM) and financial systems from manufacturers, suppliers, and distributors can retrieve information and conduct business transactions with one another. For example, a business in the market for a product could shop instantly around the globe for suppliers that meet purchase requirements and dynamically negotiate deals.
- **Counter Terrorism** – There is a pressing need in the intelligence community to provide a highly scalable system that supports collaboration, analytical reasoning and information sharing among multiple Department of Defense, intelligence and federal agencies. Furthermore, obtaining accurate and timely counter-terrorism intelligence requires processing unprecedented amounts of data, possibly in petabytes, from both classified, unclassified, structured and unstructured sources. There is no single system that can achieve this task and therefore must involve many distributed, decentralized systems.
- **Tactical Warfighting** – Similarly, in the defense sector, there is an increasing need for a C4I (Command, Control, Communications, Computers and Intelligence) system that provides a single, integrated ground picture of forces deployed to the theater. Warfighters need access to real-time information and must operate within the communications infrastructure of existing global networks. Intelligent agents, for example, may automatically discovery and correlate data streams relevant to a current tactical position. DoD’s recent Net-Centric Enterprise Services (NCES) initiative reflects this vision.

It is impossible to adopt one single platform, programming language, or protocol that fulfills the needs of these scenarios. A successful architecture must accommodate heterogeneity, and support interoperability in three dimensions: *horizontal* (across peer systems), *vertical* (among different organizational levels) and *temporal* (along a system’s evolutionary path). The unique capabilities that come with distributed Service Oriented Architectures can successfully balance these competing dimensions.

3.2 Net-Centric Enterprise Services (NCES)

Net-Centricity is an architectural mindset that values the relevance, timeliness and accessibility of information above all other qualities. A Net-Centric solution makes data immediately available to those that need it, prohibits unauthorized access to protected resources, and allows consumers to discover relevant information assets without pre-existing knowledge of their existence. The Defense Information Systems Agency (DISA)

is currently working to field a set of capabilities that help provide ubiquitous access to reliable, decision-quality information through a net-based Web-Services infrastructure.

There are currently nine Net-Centric Enterprise Services (NCES) defined, and each provides a distinct set of capabilities to the network. Infrastructure services such as Security, Storage, and Enterprise Services Management provide foundational capabilities to other services, while end-user services such as Collaboration facilitate direct communication between people in disparate locations.

With few exceptions, the services defined under NCES are platform- and implementation-agnostic specifications that abstract underlying solutions. The dichotomy formed by splitting the implementation from the specification allows COTS and GOTS implementations to appear and behave the same. That is, given a sufficiently robust specification it's possible to build adaptors to current and future technologies without impacting current integrations. From the system perspective, changes in implementation matter little because they are largely invisible. This model allows for Evolution without convolution.

Moving toward a specification-driven architecture allows for the commoditization of services defined under NCES. Achieving commoditization allows implementations to be tailored to local environments, allows deployments to be more or less robust based on expected load, and ensures that vendors compete on price, reliability and speed, not features. Net-Centricity within NCES values capabilities over implementations, and provides mechanisms that allow each member of the user community to become a catalyst of change. At the same time, Net-Centric services are reliable, fault-tolerant, secure, and provide unique capabilities that enhance both the structure and substance of the network.

3.3 Overview of XML Security Standards

This section provides a brief survey of existing security standards for XML-based messaging. The security architecture described in this document will utilize these standards, with the goal of reusing as much industry-defined work as possible. Figure 2 below illustrates the relationship and relative positioning of these standards within in the entire security "technology stack" upon which the NCES security architecture is defined. It is worth noting that these standards themselves are not silver bullets for solving every security problem. Under the "defense in depth" principle, true end-to-end application security is based upon many layers of technologies, and includes physical and network security as well as message and application level security. A truly secure system contains these levels seamlessly integrated together.

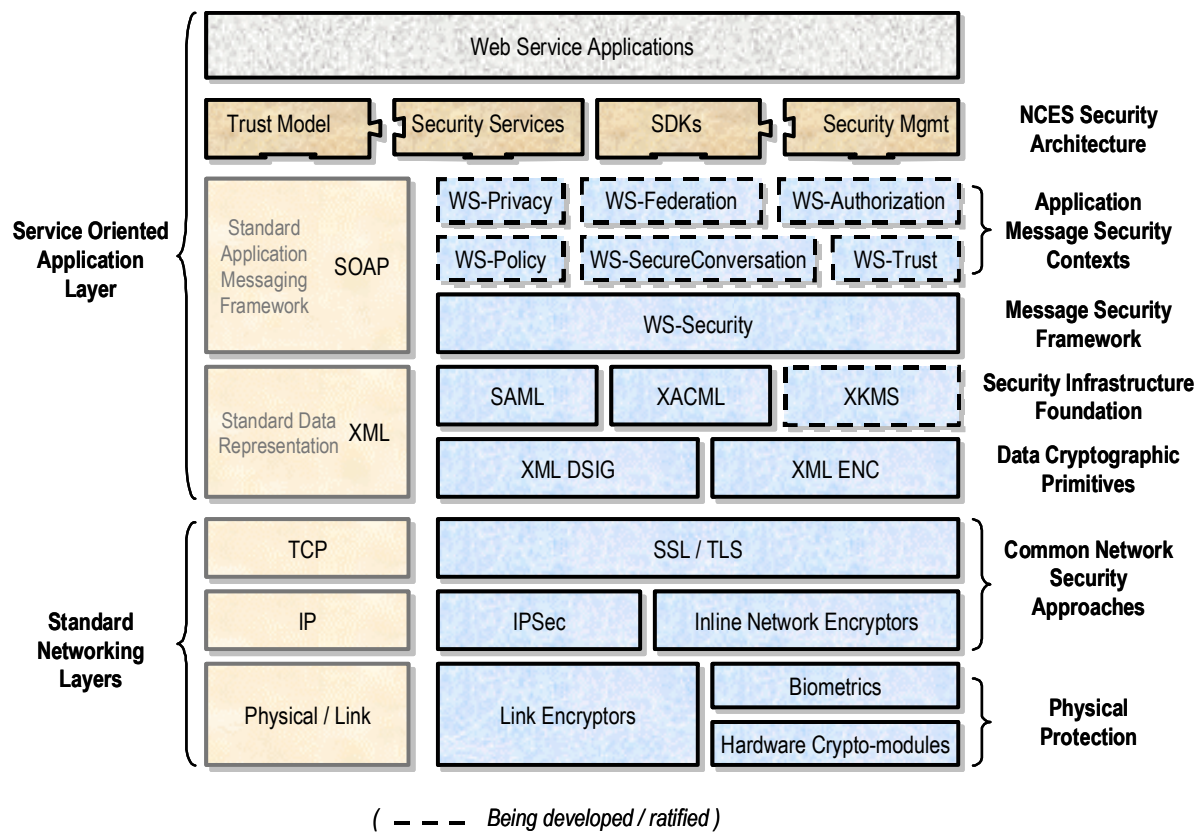


Figure 2 - Underlying Security Technology Stack

- **WS-Security**, short for Web Services Security, is a standard jointly proposed by an industry consortium (IBM, Microsoft, and Verisign) and currently being ratified by OASIS [WSS]. It serves as the foundation to address SOAP-level security issues, with three major propositions: (1) use of security tokens in SOAP headers for user identity and authentication, (2) use of XML-Signature standard for message integrity and authenticity, and (3) use of XML-Encryption for message confidentiality. There are of course many other security requirements that are not yet addressed by WS-Security. WS-Security is only the first in a series of standards proposed by the consortium aimed at providing a broader security framework for Web Services. Additional standards and vendor proposals are forthcoming that address issues such as authorization, privacy, policy, trust, secure conversation, and federation, as shown in **Error! Reference source not found.**
- **XML-Signature**. A formal Recommendation (i.e. approved standard) from W3C [XMLDSIG], this spec covers the syntax and processing of digitally signing selected elements in an XML document using either symmetric (secret) key or asymmetric (public/private) key cryptography. Such digital signatures help ensure the data integrity of the signed XML elements: any data modifications during message transit are detected in signature verification.

- **XML-Encryption [XMLENC]** is another W3C Recommendation. This spec defines the syntax and processing rules for encrypting and decrypting selected elements in an XML document for data confidentiality.
- **XML Key Management Specification (XKMS)** [XKMS] is a submission to W3C that defines a set of abstract interfaces for the underlying PKI infrastructure. The spec consists of two parts: **X-KRSS** (XML Key Registration Service Specification) that deals with public key registration and revocation, and **X-KISS** (XML Key Information Service Specification) that deals with locating and validating keys.
- **Security Assertion Markup Language (SAML).** Unlike the W3C specs above, SAML [SAML] is a standard from Organization for the Advancement of Structured Information Standards (OASIS). SAML defines a framework for exchanging security information in XML format. Security information such as authentication artifacts, authorization decisions, and subject attributes are represented in XML constructs called “assertions”, which are issued by SAML Authorities. The SAML spec also defines the protocol, transport bindings, and usage profiles for exchanging the assertions. SAML maps seamlessly to the SOAP transport, and in many areas complements the WS-Security spec (above).
- **XML Access Control Markup Language (XACML).** Ratified as an OASIS standard in February 2003 (1.0 version), XACML defines a generic authorization architecture and the constructs for expressing and exchanging access control policy information using XML. Policy constructs include policies, rules, combining algorithms, etc. XACML complements SAML so that not only policy decisions can be exchanged in a standard fashion, but policies themselves as well.

Table 1 lists the versions of the specifications supported in this architecture document:

Specification	Version
SOAP	1.1
WSDL	1.1
UDDI	2.0
WS-Interoperability	Basic Profile 1.0a
XACML	1.1
SAML	1.1
XML-DSIG	W3C Recommendation 2002-02-12
XKMS	Not yet supported
WS-Security	1.0 – SOAP Message Security 1.0 – X.509 Token Profile Draft 04 – SAML Token Profile

Table 1 - Currently Supported Standards and Their Versions

4. ARCHITECTURE OVERVIEW

4.1 Security Challenges under SOAs

The paradigm shift towards service-oriented system collaboration and composition also brings fundamental changes to the approach used to define security architectures. Most security solutions that exist today are based on the assumption that both clients and servers are located on the same physical (e.g. local LAN) or logical (e.g. VPN) network. The architectures generally rely heavily on perimeter-based security such as DMZs, firewalls, and intrusion detection to thwart security threats. Similarly, the security policies that back existing solutions are also to a large extent perimeter-based. For example, obtaining access to an application usually requires creation of a new user account on the machine or network where the application is installed, and includes granting the user physical access to the facility where the machine or network is located. By contrast, application level security is usually regarded not quite as critical as network security and oftentimes is enforced simply by a username and password.

Under a SOA, however, such perimeter-based security models are far from adequate. As Section 3.1 describes, the primary goal of a building a SOA is to facilitate Net-Centric information sharing and collaboration:

- Business functionality, previously inaccessible unless (for instance) physically sitting in front of a terminal, will become service-enabled and exposed to external consumers via standard Web Service protocols.
- Consumers – which may be services themselves – can dynamically discover services and make use of their data in real-time.
- Services are inherently location independent and not necessarily even bound to a physical location. The network addresses or “endpoints” of services are published in a service registry such as UDDI, and can change over time as services are relocated during normal system evolution or for fail-over reasons during system maintenance.
- Service consumers and providers may belong to different physical networks or even different organizations. These networks and/or organizations may be governed by entirely different security policies.

Therefore, in a Net-Centric environment, the focus on perimeter-based security models must be augmented with an application or service-level view of security. With both models in mind, the emphasis is placed not on physical ownership and control but on network identities, trust, and authorized access to resources by both users and other principals.

Security within a Net-Centric environment has its own challenges:

1. **Firewall Limitations** – Allowing inbound HTTP access to Web Services opens up servers to potential attack which may not be detectable by conventional firewall products. For example, an ill-intended SOAP message may be constructed to cause internal application buffer overflow while looking completely benign to the firewall and the HTTP server. Recently many new XML firewall products have emerged that attempt to protect Web Services at the SOAP level, but their effectiveness has not been closely studied and the positioning of those products within the entire enterprise security architecture is not yet clear.
2. **Service-Level Security Semantics** – As Section 3.3 describes, most of the standardization efforts have focused on defining the *wire formats* needed for security information exchange. The standards largely ignore the similar challenge of defining the mechanism by which different parties *interface* with each other to achieve security goals such as authentication and authorization. For example, SAML defines the XML structures and protocols for sending authentication assertions, but it doesn't prescribe who should pass what information to whom, when information should be passed, or how such information may be used.
3. **Interoperability of Security Solutions** – Because of the lack of standard profiling at the service interface level, Web Service security products in the market today are not fully interoperable even though they all claim to be compliant with Web Services security standards. Further, many are point solutions that do not meet all requirements of a DoD enterprise security architecture, and are not capable of extending beyond enterprise boundaries.
4. **Secure Composition and Orchestration** – As enterprise Web Services proliferate, there is an increasing need for multiple services to interact among one another within a joint business process or workflow. This situation presents many security challenges. For example, SOAP is not a full-blown messaging protocol and doesn't have inherent provisions for a service consumer to specify the destination(s) or the "itinerary" of an invocation sequence. As a result, the SOAP message might be replayed to unintended 3rd parties bearing the same operation signature. For more details on this issue, please see the Future Work section towards the end of this document.
5. **Multiple Security Domains and Classification Levels** – Current guard technologies are not yet connection-oriented and must evolve to support XML and SOAP message security.

6. **Security vs. Performance** – A PK-enabled security architecture involves many computation-intensive tasks such as message signing, encryption, and certificate validation. Sending a properly signed message may be many times slower than a less secure version, and there is usually a direct inverse relationship between performance and security. Cautious planning and effective optimization techniques are necessary to ensure that a secured SOA environment will meet operational requirements.

7. **Impacts on Existing Policies and Processes** – Current C&A policies generally require identification of system boundaries, whereas in an SOA based network trust relationships are established more dynamically. One possible solution is to define the C&A boundaries at the Web Service interfaces.

Defining a service-level security architecture to address these challenges is the focus of this paper.

4.2 Summary of Architectural Requirements

The primary goal of the security architecture defined in this document is to ensure Enterprise Services (ES) can be invoked securely. As with every mission critical distributed system there is a set of key security requirements that must be met:

1. **Authentication** – Most (if not all) service providers will require that consumers are authenticated before accepting a service request. Service consumers will also need to authenticate service providers when a response is received. Different authentication mechanisms should be supported, and these mechanisms should be configurable and interchangeable according to service-specific requirements.
2. **Authorization** – In addition to authentication of a service consumer, access to a service will also require the consumer to possess certain privileges. These privileges feed an authorization check that is usually based on access control policies – who can access a service and under what conditions, for example. Different models may be used for authorization, such as mandatory or role-based access control. The authorization implementation should also be extensible to allow for domain- or COI-specific customizations.
3. **Confidentiality** – Protect the underlying communication transport as well as messages or documents that are carried over the transport so that they cannot be made available to unauthorized parties. Sometimes only a fragment of the message or document (e.g. wrapped within a certain XML tag) may need to be kept confidential.

4. **Data Integrity** – Provide protection against unauthorized alteration of messages during transit.
5. **Non-repudiation** – Provide protection against false denial of involvement in a communication. Non-repudiation ensures that a sender cannot deny a message already sent, and a receiver cannot deny a message already received. This is especially important in monetary transactions and security auditing.
6. **Manageability** – The security architecture should also provide management capabilities for the above security functions. These may include, but are not limited to, credential management, user management, and access control policy management.
7. **Accountability** – This includes secure logging and auditing which is also required to support non-repudiation claims.

In addition, the following additional requirements are specific to or are also important in a SOA environment:

1. **Security Across Trust Domains** – The architecture must provide a trust model under which Web Service invocations across different trust domains can be secured, just like those within a single trust domain. All basic security requirements mentioned in Section 4.1 apply to cross-trust domain service invocations. Additionally, such invocations must be controlled by the local security policies of participating domains.
2. **Interoperability** – Interoperability is the cornerstone of SOAs, and the security architecture must preserve this to the maximum extent possible. Major security integration points in the architecture – such as those between service consumers and service providers, between service providers and the security infrastructure, and between security infrastructures in different trust domains – must have stable, consistent interfaces based on widely adopted industry and government standards. These interfaces enable each domain or organization to implement its own market-driven solution while maintaining effective interoperability.
3. **Modeling tailored constraints in security policies.** In a traditional security domain, resources and services are often protected by a uniform set of security rules that are not granular enough to meet specific application needs. Under a SOA, service provider requirements may vary in terms of how they need to be protected. For example, one service may require X.509 certificate based authentication whereas another service may only need username / password authentication. Furthermore, because clients that access a resource may or may not be from the local domain, different “strengths” of authentication and access

control may be required. Consequently, security policies must be expressive and flexible enough to be tailored according to Quality of Protection (QoP) parameters and user attributes.

4. **Allowing Integration with existing Information Assurance solutions, products, and policies.** The SOA-based security architecture does not intend to replace an existing investment in security infrastructure. On the contrary, a flexible IA solution should be designed to leverage existing IT investments without causing any redundant development efforts. Seamless integration with existing security tools and applications also increases the overall stability of the enterprise.
5. **Securing other infrastructure services within the SOA,** such as discovery, messaging, mediation, and service management.
6. **Unobtrusiveness.** The architecture should be unobtrusive to other service implementations. More specifically, to deploy unto the new security architecture, a service provider shall not have to:
 - Be constrained to use any one particular programming language;
 - Port an existing service implementation to a specific hardware platform;
 - Modify an existing implementation against any vendor-specific API interfaces;
 - Recompile or rebuild existing code sets

4.3 Scope, Assumptions, and Limitations

The following assumptions and limitations have been identified for the NCES Security Services 0.3 Release:

1. The security architecture does NOT yet cover the implementation of **Identity Management**. Rather, it aims to be flexible in this area so that it may leverage and integrate with existing and / or emerging DoD identity management systems.
2. The security architecture will support integration of digital certificates issued from the DoD PKI. For this release, the architecture will be based on explicit trust of the Certification Validation Service, which serves as a domain's **trust anchor** for establishing the authenticity and validity of certificates. Please refer to Section 10.2 for detailed discussions on this approach and its alternatives.
3. The architecture currently does NOT yet address **"edge" security** such as end-user authentication or end-user Single Sign On (SSO). The ultimate SSO experience, from the end user's perspective, involves authenticating not just to Web Services, but also to the network, the operating system, and/or any Identity

Management system in use in the organization. This is beyond the scope of this document and will be addressed in the near future.

4. The architecture currently does NOT support establishment and protection of **security contexts** that span across multiple Web Services.
5. The architecture currently does NOT address message level security across multiple security levels (MSL) or in a **multi-level security** (MLS) environment.
6. This architecture does NOT yet define **enterprise audit** and logging functionality and related service specifications. They will be provided in the near future.
7. The architecture currently does NOT directly address content or **data level access control**, such as enforcing proper access control over data contents and information products contained as a SOAP message “payload”. For now this is considered the responsibility of the service provider. However, the NCES Security Services do provide support for managing and accessing security policies that may be leveraged for this task.
8. The architecture assumes sufficient protection of physical **security infrastructure components**. Security services such as the policy and credential management services (as will be introduced in Section 4.5), as well as security repositories such as the policy store should be protected using well-established policies and practices for securing access to physical systems.
9. Web Services and the security infrastructure components (e.g., Policy Decision Points) should also be sufficiently protected from **format and data attacks**. Some of these attacks may be addressed by existing COTS products such as XML firewalls, while others involve conformance to good security programming practices such as preventing buffer overflows. The architecture document does not yet address the means of performing these functions.

4.4 Conceptual Enterprise Security Architecture

Figure 3 presents a very high level illustration of the security architecture. The diagram reflects the following concepts:

1. Service consumers and providers (shown on the left side of the diagram) exchange security related information (e.g. certificates) with each other through open security standards such as WS-Security and SAML.
2. The underlying security infrastructure is exposed as Web Services (shown in the middle section of the diagram). This document defines a set of *Security CES*

using technology-agnostic WSDL interfaces. Security functionality such as credential and policy management are themselves wrapped as Web Services.

3. Instead of attempting to implement all security infrastructure components from scratch, the security services leverage existing enterprise security infrastructure (shown on the right side of the diagram) such as identity management, and PKI through an integration backplane.

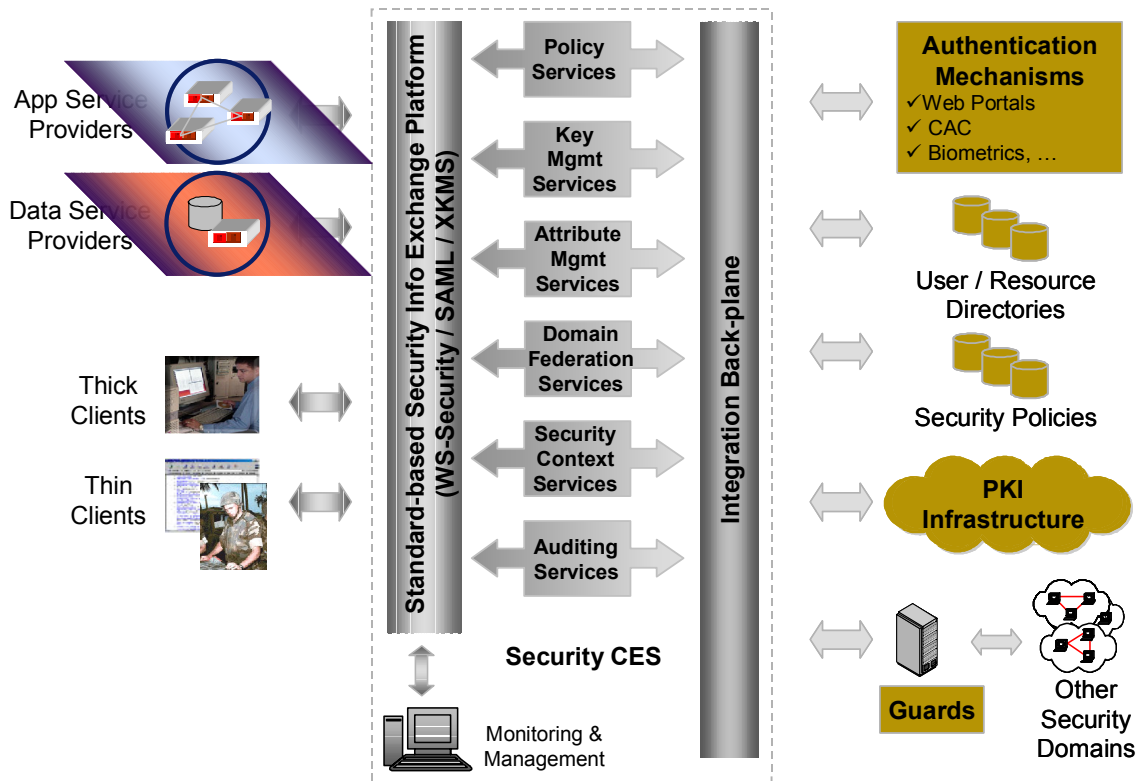


Figure 3 - Conceptual Enterprise Security Architecture

The security architecture provides many important benefits:














- **Efficiency.** A set of security services along with their backend infrastructures are responsible for supporting application level security, so that service providers themselves do not have to roll their own enterprise security management. This is not only efficient but also significantly reduces operational overhead. (unless the provider has application-specific security requirements which can be built on top of the Security CESs).
- **Plug-and-play.** The security service abstraction layer is able to provide true loose coupling among applications and enhance overall system stability. As can be seen from the diagram, this is reflected in the plug and play capability for both security users and security infrastructure providers: On the left hand side of Figure 3, service providers and consumers can easily plug in to the security

framework because all interfaces are fully standards based. The right hand side of the diagram details how security developers can swap security implementations without affecting the Web Services and end users. These changes are possible because the security service interfaces in the middle remain the same.

- **Cross-domain interoperability.** The security architecture is standards-based, which enables secure collaboration and information sharing across trust domains. As long as all trust domains conform to the same set of security service specifications they are able to exchange security claims, entity attributes, and access control policies. This level of cross-domain interoperability is critical for a Net-Centric environment.
- **Future-proof.** The architecture is well positioned for future evolution. The security service specifications are platform-independent, technology-agnostic, and vendor-neutral. This not only promotes reuse of existing infrastructures but also allows for market-driven solutions and selection of best-of-breed products without committing to vendor lock-in.

4.5 The Security CESs

The Security CESs, depicted in the middle box of Figure 3, consist of a number of functional **service groups**, each of which may include one or more service interfaces that perform specific tasks. In the current release, only the very basic interfaces that support core Web Service security capabilities are defined; more services will be added and existing services will be expanded in future versions to provide richer and more sophisticated features. The taxonomy or functional breakdown of the Security CES is shown in the table below:

Service Group	Current Services	Future Services*
Policy Services	 Policy Decision Service  Policy Retrieval Service  Policy Administration Service	 Policy Subscription Service
Credential Management Services	 Certificate Validation Service	 Certificate Registration Service  Certificate Retrieval Service
Attribute Services	 Principal Attribute Service	 Resource Attribute Service  Environment Attribute Service
Trust Domain Federation Services	None	 Domain Inquiry Service  Domain Registration Service
Security Context Services	None	 Security Context Service


Service Group	Current Services	Future Services*
Auditing and Logging Services	None	 Secure Logging Service  Auditing Service

Table 2 - Security CES Taxonomy

** Note the future service offerings and their names are subject to change*

In the following sub-sections, the service groups and their current member services are briefly described. Detailed technical specifications of the above services will be defined in a separate document.

4.5.1 Policy Services

This service group provides policy-based authorization and access control for Web Services and system resources. The current services include:

- **Policy Decision Service** – serves as a SAML authorization authority for service providers that choose to use an external Policy Decision Point (PDP). This service accepts authorization queries and returns authorization decision assertions, all of which conform to the SAML Protocol. The heart of the service is a policy evaluation engine, which applies policies based on a variety of inputs such as the target resource, the action or operation requested, identity of the requester, etc.
- **Policy Retrieval Service** – exposes security policies in XACML format. This service can allow service providers to retrieve policies for their resources, especially when they choose to implement their own PDP logic (see Section 8.2 for details). This service can also be used by applications other than Web Services to retrieve stored resource policies (e.g. access control over portlets in a portal server).
- **Policy Administration Service** – This service uses XACML as a standard policy exchange format and can be used by management applications to compose, modify, and control authorization policies. Depending on the access control model adopted in the domain, this service's functionality may include Create, Read, Update, Delete (CRUD) operations for policy rules, rule sets, roles, permissions, security categories and compartment labels, among others. Section 9 describes the recommended RBAC model upon which the current reference implementation is based.

In the near future, a **Policy Subscription Service** will also be defined, along with related callback interfaces that allow interested parties to subscribe to and thereby receive real-time notifications on policy changes.

4.5.2 Credential Management Services

This group of services provides access to the underlying DoD PKI infrastructure. The group is envisioned to offer a subset of XKMS functionality, starting with the X-KISS spec and potentially moving into X-KRSS as well. Currently only the following service is defined:

- **Certificate Validation Service (CVS)** – This service allows clients to delegate part or all certificate validation tasks, which is especially useful when the client side doesn't have the capability for PKI processing. The service corresponds to a "Tier 2 Validation Service" as defined in the XKMS spec, and shields client applications from such PKI complexities as X.509v3 certificate syntax processing (e.g. expiration), revocation status checking, and certificate path validation. Moreover, offloading validation operations allows it to be done in a more efficient and optimized fashion. The architecture allows a service provider to use the Certificate Validation Service (CVS) to only perform revocation status checking, in which case the CVS functions similar to a PKIX Online Certificate Status Protocol (OCSP) responder; OR to offload the entire certification validation to CVS. This service, as well as the underlying DoD PKI functionality, is discussed in much greater detail in Section 10.2.

In the future the following additional services could potentially be provided:

- **Certificate Registration Service.** Web Service providers and consumers require public key certificates to perform digital signature and encryption operations with other clients. If a client generates its own public/private key pair, it needs to request a certificate for the public key from the DoD PKI. The Certificate Registration Service would use the XKMS XML Key Registration Service Specification (X-KRSS) "register" service as the interface presented to Web Service clients for public key certificate request and response. The X-KRSS "register request" message contains the identity and associated public key of the certificate being requested, and a proof of possession element (i.e., proof that the certificate requester is the actual holder of the private key corresponding to the public key in the request). The Certificate Registration Service translates the information contained in the X-KRSS register request to the certificate request format(s) used by the DoD PKI. The X-KRSS "register response" message contains the resultant X.509 public key certificate.
- **Certificate Retrieval Service.** Web Service providers and consumers must have the capability to obtain the public key certificates of users and other Web Services clients for the purposes of authentication verification, digital signature verification, and public key encryption operations. In some cases, the originator's public key certificates are included in XML messages inbound to the

Web Services client. In other cases the certificates are not included in the inbound messages and the Web Service clients must retrieve them from the DoD PKI directory system. The Certificate Retrieval Service would use the XKMS XML Key Information Service Specification (X-KISS) “locate” service as the interface presented to Web Services clients for public key certificate retrieval. The X-KISS “locate request” message contains the identity of the public key certificate being sought. The Certificate Retrieval Service interfaces with a DoD PKI directory system (e.g., the DoD Global Directory Service (GDS)), to locate and retrieve the requested certificate. The X-KISS “locate response” message contains the requested X.509 certificate, if it is found in the DoD PKI directory system. Note that the Certificate Retrieval Service would not check the revocation status or validity of the certificate retrieved; it would simply return a certificate if one is successfully located in the directory.

4.5.3 Attribute Services

In order to support policy-based decisions, various attributes are needed. This includes those of the principals, the system resources, and the application environment. This service group provides standard access mechanisms for such attributes, and defines how attribute queries are returned as SAML attribute assertions. The request-response mechanism is also based on the standard SAML Protocol. A Principal Attribute Service is currently defined:

- **Principal Attribute Service** – provides query and retrieval interfaces to access attributes for principals, which may be individuals or even organizations. The attribute taxonomy or “schema” is not defined by the service, but rather by the underlying attribute authorities (e.g. identity stores). These attributes are retrieved upon request and provided as SAML assertions that may be used as inputs to the policy decision logic. Currently, principal attributes are primarily managed by existing Identity Management systems and then stored in various directories. Therefore this service provides just the “read” functionality to obtain those stored attributes in an assertion that binds them with the principal’s identity.

In future versions of this service, additional sources of principal attributes will be considered, and mechanisms will be provided to restrict access to sensitive attributes.

In the near future, a **Resource Attribute Service** and potentially an **Environment Attribute Service** will also be defined for retrieving resource and environment attributes, respectively. When it becomes necessary to actively manage those attributes, a set of Attribute Administration Services providing the complete CRUD operations will be defined as well.

4.5.4 Trust Domain Federation Services

The Trust Domain Federation Services is responsible for managing a trust domain's trust relationships with other domains. Its interfaces may include registering and deregistering other domains as trusted parties, and inquiring about established trust relationships.

These services will be defined in future versions of the architecture.

4.5.5 Security Context Services

These services provide mechanisms for sharing *security contexts* across multiple Web Services. Such contexts are necessary in a dynamic SOA environment where indirect or brokered trust relationships abound. For example, when service A invokes service B on an end-user's behalf (see Section 6 for detail on the service chaining scenario), a common security context can help establish a boundary for this unit of work so that, for instance, service A cannot forward the request to an unintended service C even if A possesses a valid user assertion.

In an enterprise service environment, security contexts are important in addressing service orchestrations and workflows. They may also help improve efficiency especially in interactive scenarios. For example, results of certain authentication and authorization steps may be performed only once for a series of consumer-provider interactions within a common security context.

The Security Context Services will be defined in the future. Currently there are standard proposals such as WS-Trust, WS-SecureConversation, and WS-Coordination that address this topic. Future service specifications will consider conformance to them once they become approved.

4.5.6 Auditing and Logging Services

Enterprise auditing is also an important requirement for the security architecture. Two pieces of functionality need to be provided: recording the service level activities (logging), and identifying anomalies (such as access violations or attacks) from those records. Currently the service level logging are performed locally by SDKs (see Section 6.1 for details) deployed at service consumers and providers. The logs include:

- Outbound message information (message ID, sending timestamp, host, target service, etc.)
- Inbound message information (message ID, receiving timestamp, etc.)
- Message signature verification (success / faults)

- Certificate validation and status checking results (success / faults)
- Policy decision results (permit / deny / indeterminate)
- Invocation status (resource, action, success / faults)

In the near future, service interfaces will be defined for remote logging and auditing.

To reiterate, the service interfaces defined by this architecture are *specifications*, not implementations. The actual implementations may utilize best-of-breed COTS and GOTS technologies and may vary in different IT environments, but the specifications will remain stable and interoperable. Going forward it is envisioned that the specifications will be driven by the collective efforts of various NCES initiatives and their requirements, while at the same time reflecting current industry best practices.

5. THE TRUST MODEL

5.1 The Micro View

The security architecture defined in this document consists of two logical components:

- 1) A trust model under which Web Service consumers and providers interact with one another;
- 2) A set of security Core Enterprise Services that provide the functionality necessary to support the model and are based on open XML security standards. These services were defined in the previous section.

This section introduces the basic trust model that serves as the foundation of the NCES security architecture. It is presented in a “bottom-up” approach: We start with the “micro” view by looking at the trust model and associated Security Services in a single Trust Domain; then we zoom out to the “macro” view, describing how the model and supporting services would look when multiple Trust Domains are involved.

Under the basic model, a Trust Domain consists of a “triad” of one or more **Web Service Providers**, **Service Consumers**, and a set of **Security CESs**, as shown in Figure 4 below. As defined previously, a Web Service Provider may provide a collection of multiple Web Services, hosted together for a common business purpose. The provider may be physically located on one or more server machines. The diagram shows two kinds of Web Service consumers:

- *“Edge” applications*, which provide the presentation layer and user interaction logic (often web-based, but does not have to be), and serve as entry points of end users for accessing the service layer. The edge applications initiate service requests on behalf of end users. As mentioned in the assumptions, this service level security architecture does not focus on the security between individual users and edge applications. The mechanism, context and strength of edge security, however, must be captured and propagated for downstream policy decisions, as discussed in later sections.
- In addition, a service provider may in turn be a consumer of other Web Services. This is often seen in service composition and workflow scenarios.

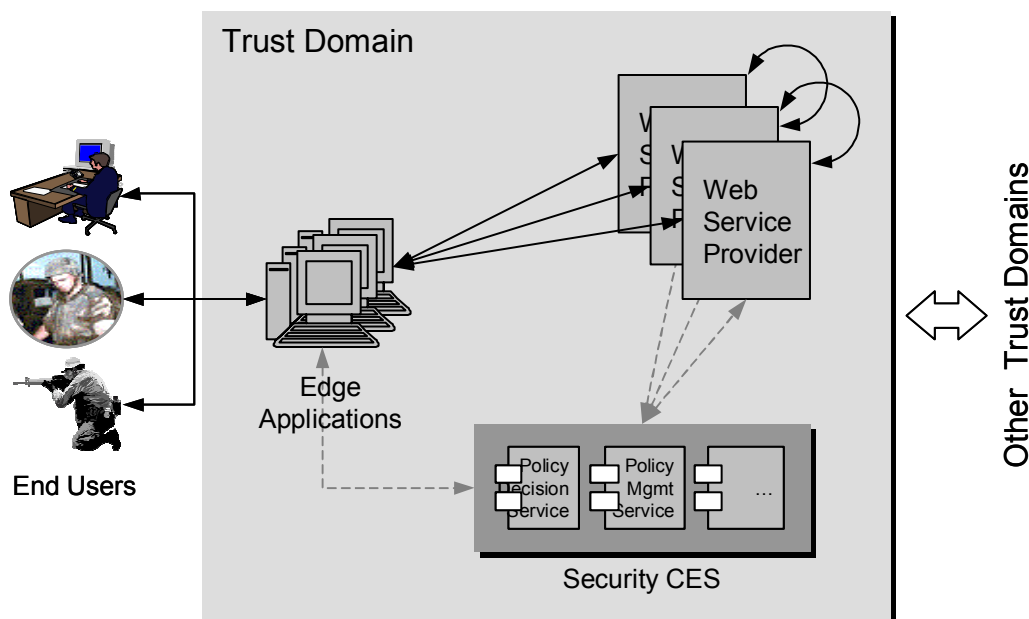


Figure 4 - The Micro View: Within a Single Trust Domain

This diagram reflects the concept of *Indirect*, or *Brokered Trust*: one party trusts a second party who, in turn, trusts or vouches for, a third party [WST]. Furthermore, instead of having Web Service providers themselves authorize users and control their access (i.e. the *Direct Trust* model), such responsibilities are assisted and sometimes taken over by the Security infrastructure and related CESs.

The brokered trust model has some obvious benefits when compared with the direct trust model:

- It's non-invasive because service providers do not have to bear the burden of implementing security enforcement logic and managing security policies;
- Service providers can, on the end user's behalf, serve as active intermediaries or "brokers" of information offered by other services, thereby enabling "power to the edge".
- Because brokered trust relationships are more flexible and more dynamic, the architecture is more resilient and able to quickly adapt to new missions and new business requirements, making it well suited for the dynamic Net-Centric environment.

Section 6 describes this security model within a single trust domain in much greater technical detail.

5.2 The Macro View

Earlier in the document a Trust Domain was defined as a purely logical realm that contains a set of system entities that are governed by a set of common security policies. The delineation of a (logical) trust domain boundary really depends on the extent of the policies; it *may or may not* coincide with organizational network boundaries (e.g. security enclaves). Just as policies may be defined at different organizational levels, trust domains may also be created at different levels or scales. In addition to the **DoD Enterprise Domain** that is the primary focus of this architecture, there may also be smaller domains that could benefit from an approach such as this. Examples of such domains include:

- A *local trust domain* containing resources and services controlled under the discretion of an individual user, located on the user's PC;
- An *enclave-wide trust domain* containing resources and services in a security enclave;
- An *agency-wide trust domain* that coincides with the administrative domain of a DoD agency or other type of office. For example, the Defense Information Systems Agency (DISA) may have its own trust domain;
- A *DoD Enterprise domain* for all DoD Enterprise Services;
- A *Community of Interest (COI) domain* that encompasses multiple organizations collaborating with one another sharing a common interest. Examples may include an Intelligence Community (IC) domain, a Command and Control (C2) domain, and so on. A COI trust domain might in fact overlap with other trust domains.

From the "macro" view, the architecture can be summarized (rather informally) as follows:

- 1) The security architecture consists of multiple associated trust domains at multiple levels. A trust domain may have peers and /or may join a "parent" domain;
- 2) Within a trust domain, the local security policies control access to local resources by both local principals and principals from other trust domains;
- 3) Trust domains may overlap one another. In this case, a resource may be governed by different policies from more than one domain depending on the service invocation context.

The model is conceptually depicted in the diagram in Figure 5.

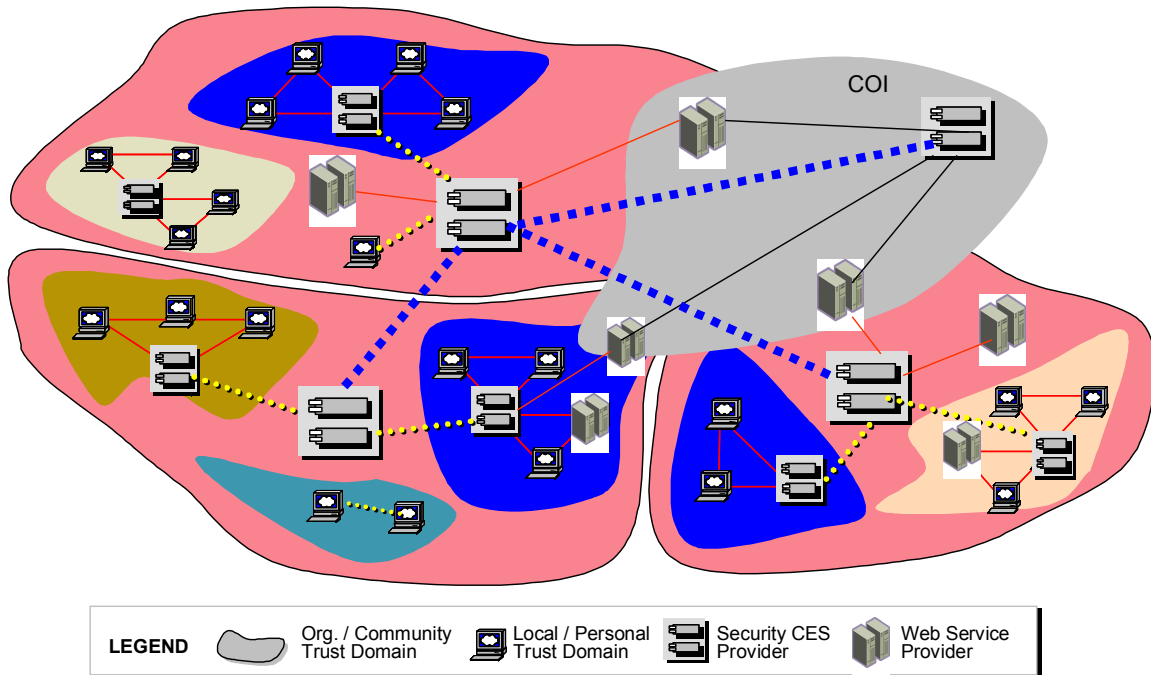


Figure 5 - The Macro View: Across Trust Domain Boundaries

Although the diagram doesn't reveal many of the technical details, it does manifest some of the important design principles outlined for cross-trust domain security and trust:

- **Decentralized trust domains** - In a large-scale distributed computing environment such as the DoD Enterprise, it is extremely difficult to set up a single centralized security management system. Although enterprise security is important and necessary, it can become too resource-intensive to build and maintain if it tries to satisfy all the non-enterprise information assurance needs. These local-, agency-, or COI-level needs tend to vary significantly according to different mission requirements. Furthermore, when it comes to collaboration involving multiple autonomous organizations, creating a centralized system is simply not possible because each organization may be required to oversee certain information assets under existing laws and regulations. In light of this, the security architecture defined in this paper proposes a trust model that is enterprise-strength and yet decentralized based on network identities and brokered trust.
- **Support for Peer-to-Peer (P2P) horizontal integration** - With dramatic advances in hardware technologies, more and more computing power is now "on the edge", and users are increasingly becoming both consumers and producers of data. Consequently, the proposed architecture allows service-level peer-to-peer interactions across trust domains (in contrast to a "hub-and-spoke" model) to facilitate horizontal information integration under the Task, Post, Process, Use

(TPPU) paradigm. For example, several Web Services from different organizations may join a COI and work together in enhancing target tracking data for better situational awareness. Note that such interactions will of course be subject to enterprise security policies as well as domain-specific constraints.

- **Delegation of authority for increased manageability** – In addition to horizontal integration, the architecture also supports seamless vertical integration. As already mentioned, in the DoD Enterprise with potentially millions of users and millions more resources, having a central authorization authority is too inflexible and simply won't scale. With a model of multi-level trust domains, policy management on the resources is delegated down to the appropriate level where the resource is located and owned, from communities to organizations or even to individuals, effectively optimizing the management span at each level. The enterprise domain can then focus on managing critical, enterprise-wide resources and services, while leaving other entities to be managed by lower-level domains. The delegation happens in the other direction as well: policy decisions that are beyond the discretion of the local trust domain (e.g. granting access to a principal from an unrecognized trust domain), can be passed on to other authorities (e.g. the parent trust domain or a COI authority).
- **Self-similarity** – As shown in Figure 5, at different scales, the trust domains interact with one another in basically the same fashion. In fact, the same set of NCES Security Services are used at every domain level. In other words, the architecture exhibits "self-similar" (a.k.a. "fractal") behavior, which bears this important characteristic: *To design efficient models for services at various scales of the network, it is often sufficient to understand the behavior and characteristics at a fairly simple scale level.* The benefit of this behavior is therefore two-pronged: it gives the architecture infinite scalability yet also contains its complexity. It has been argued that the success of the World Wide Web is in part due to the pervasive self-similarity exhibited in its usage behavior [SIMWEB].

In this release a focus is placed on the intra-trust domain aspects of the security architecture. Technical details of the cross-trust domain aspects will be covered in future versions of this document.

6. SECURITY ARCHITECTURE WITHIN A TRUST DOMAIN

The trust model introduced in the previous section was purposely technology-agnostic: There was no mention of any specific wire protocols, interfaces, or data models, except that everything will be Web Services based. Starting in this section we begin filling in the semantic and syntactical details, taking advantage of industry standards from the technology stack in Figure 2 as much as possible. Over time, these standards will evolve and new standards will continue to emerge, but the conceptual architecture is inherently stable and will remain the same.

6.1 Securing the Invocation Path

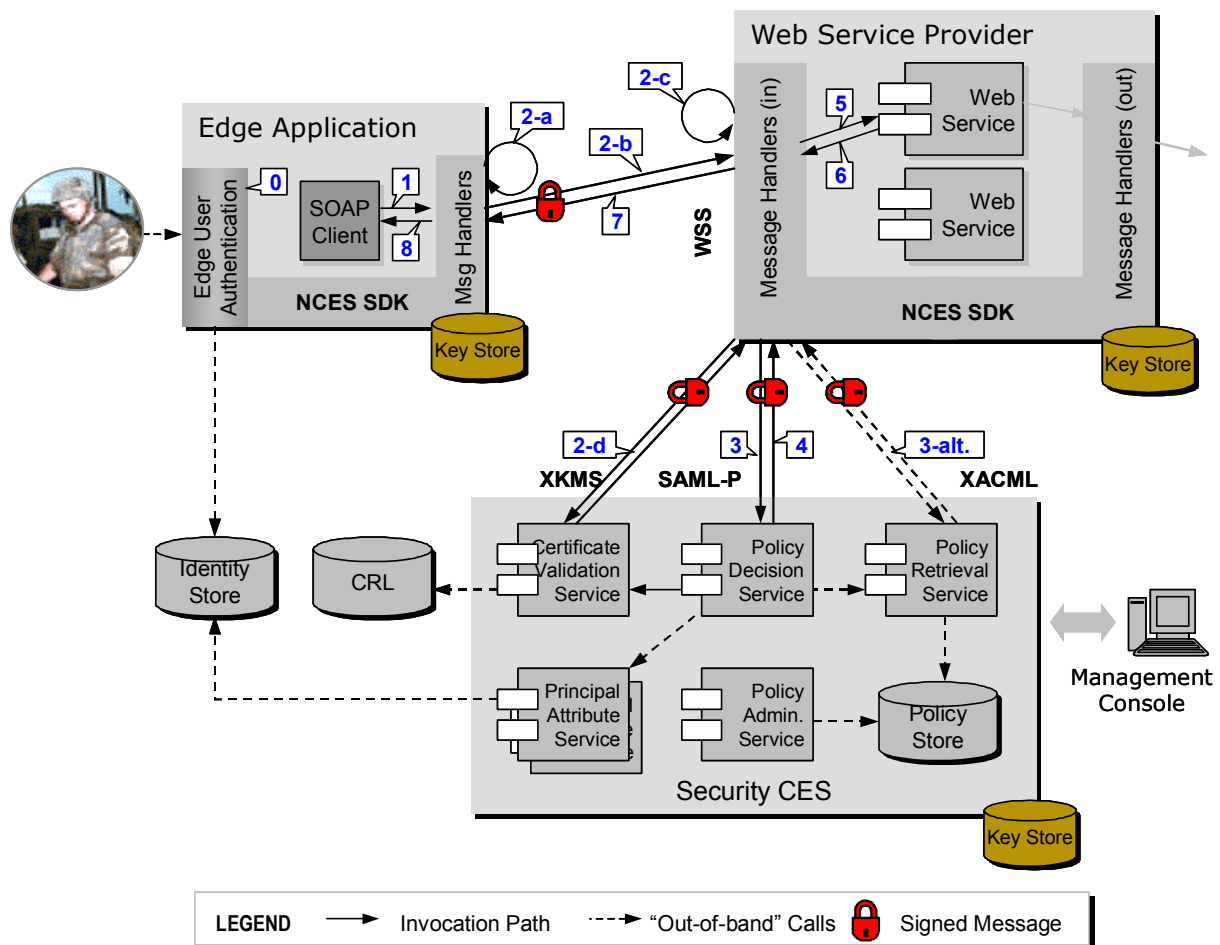


Figure 6 - Logical Security Architecture, Single Trust Domain

Figure 6 illustrates the logical security architecture using a typical scenario involving an edge application (e.g. a web portal) acting as a service consumer that invokes and consumes a Web Service provider.

In addition to the Security CESs, the diagram introduces some additional components involved in the security architecture:

- For service providers and consumers, an NCES **Service Development Kit (SDK)** is typically deployed along with the application to facilitate interactions with NCES Core Enterprise Services (which includes security). A *reference implementation* of the SDK is being provided along with the Security CES, but additional implementations that leverage the same architecture may be built in the future.

It is important to note that SDKs are platform- and technology-specific tools that are only provided as developer aids, so that access to the standard Security CES interfaces may be rapidly enabled. They are not an architecture component per se, nor should they be treated as standards. To achieve true Net-Centricity, applications should only rely on the Web Service standards, Security CES WSDL interfaces, and the processing rules as outlined in this document. In fact, it is perfectly possible for a service provider to implement its own security logic that is compliant to the security architecture without using a third-party SDK.

- The security part of the SDK also includes a set of **SOAP Message Handlers**, which can “intercept” inbound and outbound* SOAP messages at runtime and apply security related processing in a way that is transparent to application logic. Multiple message handlers can be “chained” together and can be configured at deployment time, which enables flexible and extensible security configurations. For example, an auditing handler may be added in front of the authentication and policy enforcement handlers, as illustrated in Figure 7:

* Note that “inbound” and “outbound” are relative terms. The same SOAP request is an outbound message to the service consumer but an inbound message to the service provider. Because a service provider can also be a consumer to other services, the SDK includes both inbound and outbound message handlers.

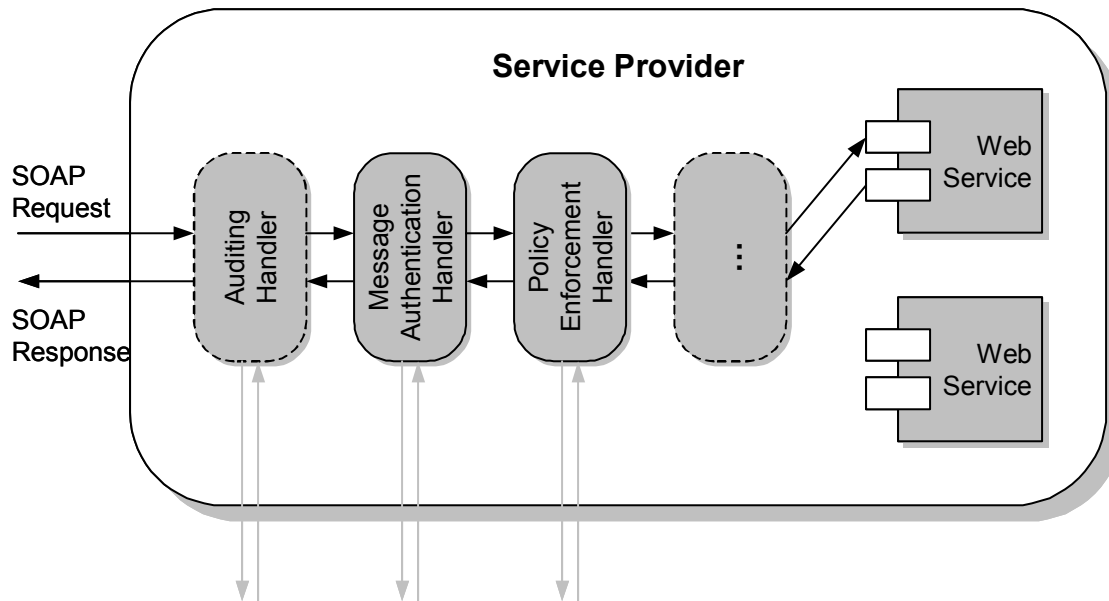


Figure 7 - Chaining of Inbound Message Handlers

Note that, although the message handler notion coincides with those in JAX-RPC based toolkits, we treat it here as a pure logical concept, not an implementation technique. Depending on the physical system architecture, the handler may map to a software component within the Web Service runtime, or a standalone application by itself.

- The SDK may also leverage a **key store** for PKI based message signing and validation purposes. The key store is used to store the private key(s) and trusted anchor certificates (e.g. DoD PKI CAs and CVS certificates). The SDK can be configured to work with different key stores (software or hardware) and different underlying cryptographic providers.

Figure 7 also shows the logical sequence of a properly secured Web Service invocation. After the user authenticated, the edge application initiates the Web Service request on the user's behalf. The service invocation involves the following steps, corresponding to the numbers marked in Figure 6:

- Step 0. The end user is authenticated, and a SAML assertion is produced by the edge application*. See **section 7.1** for more details of this step.
- Step 1. The edge application issues a Web Service request in the form of a regular SOAP message. The message is intercepted by an outbound message handler in the NCES SDK;

* Note that in reality, Step 0 i.e. edge user authentication may have happened some time before the initiation of the Web Service request and that the user is often given a valid session for a limited period. In this case it is recommended that a SAML assertion construct is generated per user session and cached for use by all service requests initiated from this session, however this is an implementation issue that is up to the edge application.

Step 2. Signed SOAP request is sent from client to server, using WSS and XML-DSIG protocols as prescribed in Section 7.2. This step can be further broken down into the following:

2-a. The outbound message handler on the client-side signs the SOAP request using the application's private key. The digital signature not only serves as the "proof of possession" of the private key which ensures message authenticity, but also protects the message integrity so that any tampering in transit would be detected;

2-b. The SOAP request is sent to the target service provider using standard Internet protocols (e.g. HTTP);

2-c. An inbound message handler on the server side intercepts the SOAP request, then verifies the digital signature contained in the message. The handler also validates the sender's PKI certificate to make sure it's authentic*;

2-d. The message handler further checks the status of the sender's certificate against the Certificate Validation Service, making sure the certificate is not revoked.

In addition to the above steps, the handler in fact performs additional checks such as detecting replay. **Section 7.2** describes message level authentication in greater detail. The above steps are common for all service requests and responses that need authentication, such as Steps 3, 4, and 7 in Figure 6. For simplicity sake, the sub-steps a ~ d are not repeated in the diagram for the latter messages.

Step 3. After successfully authenticating the request, the inbound message handler proceeds to authorize the request. It MAY send a SAML authorization decision query to the Policy Decision Service (PDS), passing the resource identifier, the intended action, and supporting evidence such as the SAML assertion from the SOAP request. This message is also signed using the service provider's private key for message authenticity and integrity protection.

Step 4. The PDS evaluates the SAML query and makes a decision (permit / deny / indeterminate) based on the evidence and the access control policies of this trust domain. If necessary, the PDS may choose to obtain additional attributes for the user and the resource involved. The decision is sent back to the service provider in the form of a SAML authorization decision assertion within a signed SOAP response.

* As mentioned before, in the future the handler may also choose to offload the validation step to the CVS in Step 2-d, but in this release CVS only performs revocation status checking.

Alternatively, if the service provider chooses to make the policy decision by itself, steps 3 and 4 are no longer necessary. In support of this, the SDK may request relevant XACML policies from the Policy Retrieval Service (PRS), shown as step “3-alt.” in the diagram. Further, policy retrieval can happen in advance and therefore does not have to be in the invocation path.

It is worth noting, however, that the decision whether or not to use an external PDS may also depend on administration policies of the trust domain.

Section 8 formally describes the policy based authorization architecture.

Step 5. If access is authorized, the message handler forwards the original SOAP request to the target Web Service for processing.

Step 6. The service processes the request and sends back a synchronous SOAP response.

Step 7. When the response goes back through the message handlers, it is signed with the provider’s private key using identical steps shown for Step 2.

Step 8. On the consumer side, after verifying the provider’s signature, the message handler passes the SOAP response to the edge application.

6.2 Service Chaining*

This basic invocation sequence may be easily extended to more complex usage scenarios. For instance, the Web Service may invoke another service on the user’s behalf. This so-called “service chaining” scenario is shown in Figure 8 below.

* Note that this document differentiates between “Service Chaining” and “Service Routing”. The latter refers to a Web Service request being forwarded through intermediaries, intact, to the target service.

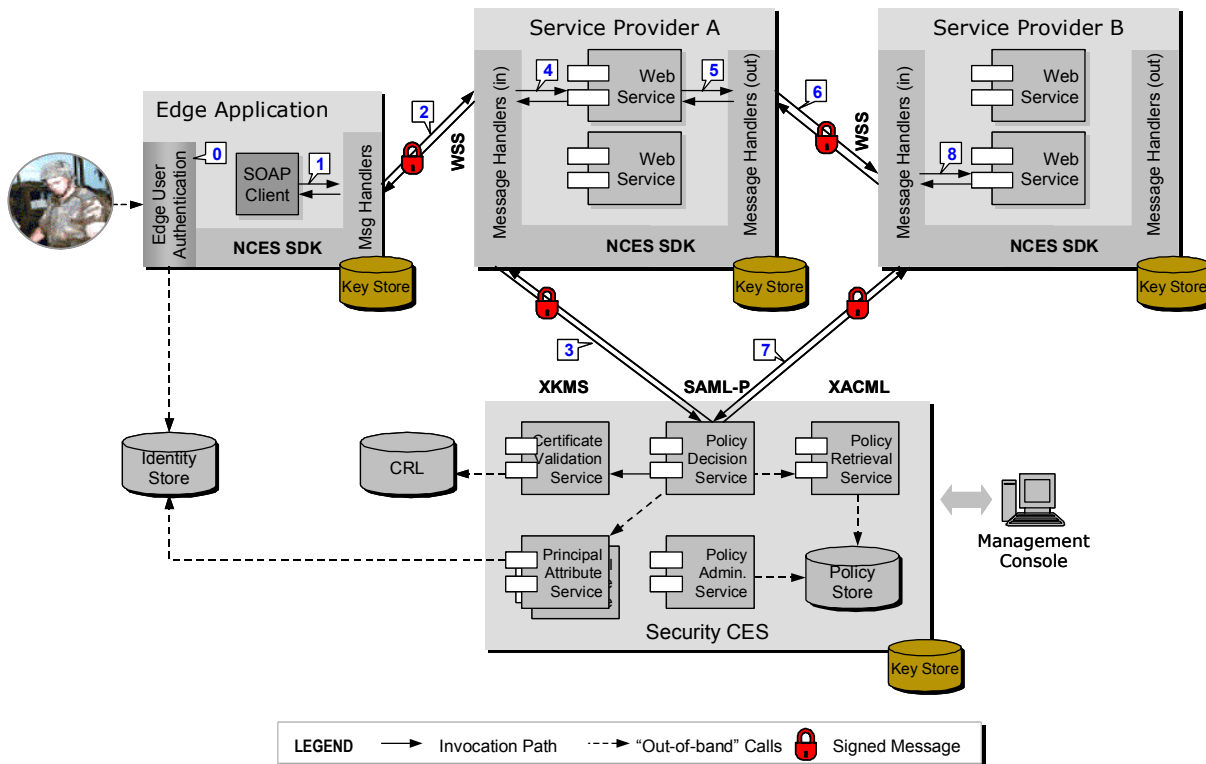


Figure 8 - Service Chaining

In this case, Service Provider A is also a consumer of Service Provider B, and acts as a SOAP client just like the edge application. The invocation steps in the diagram (Steps 0 to 8, response path not numbered) are self-explanatory.

From the authentication perspective, when Service Provider A makes a request to Provider B, two scenarios are possible:

- In many cases, Provider A invokes B **on behalf of the original end user**. This would require Provider A to pass on the *original* SAML assertion in the second SOAP request, as opposed to creating a new one. Here Provider A “vouches for” the assertion by signing the message with its digital signature, even though it is not the issuer of the assertion. In the same way that Provider A authorized the first request, Provider B then needs to authorize the second request based on both its policies applicable to the end user AND its trust relationship with A.
- Alternatively, Provider A may choose to invoke B **on its own behalf**. This is also not uncommon. For example, B may be a data service that provides “wholesale” data to a small handful of applications, and may be relying on them to deal with end user related access control policies. In this case, Provider A still needs to authenticate itself to B (e.g. via digital signatures), but no longer needs to pass on the originating user assertion. Provider B’s authorization decision will be based on A’s principal instead of the original user’s.

1108

1109 Both scenarios are supported under this architecture. The next two sections will
1110 examine them in much greater detail.

1111

1112

7. AUTHENTICATION

Because the subsequent sections involve more technical discussions in the SOAP messaging layer, it is necessary to adopt a set of consistent messaging terms so that their connotations can be clearly understood within the context of this document. In addition to Service Consumer and Service Provider, the following terms will be used:

- Message *sender* and *recipient*. These terms are used in a point-to-point context and are only relevant to a particular SOAP message (either request or response). For example, a service consumer is a sender of a SOAP request message but a recipient of a response message.
- Message *originator*. As discussed in Section 6.2, a Web Service request may be issued on behalf another system entity, either an end user or an application. This entity, if exists, is called the originator of the Web Service request of interest.

These terms are illustrated in Figure 9 below:

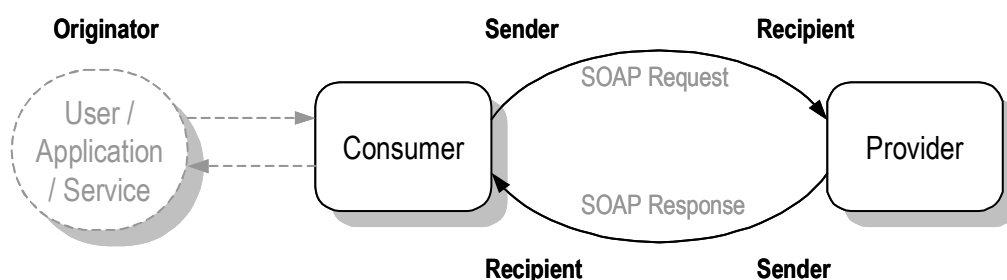


Figure 9 - Messaging Terms

7.1 Asserting the Authentication of End Users

In the scenario illustrated in Figure 6, the edge application is responsible for authenticating end users, which is shown as “**Step 0**” in the diagram. The exact mechanism and policies for this step depend on the actual portal server as well as the trust domain environment, therefore is not prescribed here. However, in order for downstream service providers to make authorization decisions, the user identity and the actual context under which the user is authenticated need to be propagated along with the Web Service request.

In order to access downstream Web Services on an end user’s behalf, the edge application **MUST** provide a SAML assertion that includes the following information:

1. [REQUIRED] Issuer of the assertion is required by the SAML spec, which is the identity of the edge application itself;
2. [REQUIRED] An assertion ID as required by the SAML spec;
3. [REQUIRED] One Authentication Statement containing all of the following:
 - Timestamp of the authentication instant;
 - The authentication method (e.g. Client SSL, see the SAML spec for the exact URI values. Additional values will be defined later for DoD enterprise use);
 - The <Subject> element containing the user identifier, which includes both the naming scheme (e.g. "X.509SubjectName") and the identifier (e.g. "CN=John Doe, OU=NCES, DC=DISA, DC=MIL");
 - A <SubjectConfirmation> element within the <Subject> element that MUST use the "sender-vouches" confirmation method as defined in the WS-Security SAML Token Profile [WSS-SAML]. Given our brokered trust model, the sender of the SOAP request is not necessarily the issuer of the assertion; therefore we can only trust the sender to vouch for the assertion statements.
4. [OPTIONAL] Zero or more Attribute Statements describing any additional authentication context, in addition to the authentication method URI, that may be relevant to downstream access control decisions. For example, if the portal server is temporarily unable to check the user's X.509 certificate status against the CRL but allows the user to log in regardless, such information should be made available to target service providers. Such attributes, if any, should be officially defined at the DoD enterprise level.
5. [OPTIONAL] Zero or more Attribute Statements for additional user attributes (e.g. contact info, roles). *Due to confidentiality and privacy concerns, such attributes should be "public" attributes of the user that are visible to all parties in the trust domain. Definition of such attributes is the responsibility of the trust domain (see also the Future Work section).*

An example of the assertion is shown in Appendix A.1. This assertion will be passed in the SOAP request message indicating that the request is made on the user's behalf.

The edge application MUST NOT sign the assertion alone, but rather sign it along with other elements in the request message including the SOAP body (see Section 7.2.1 for details). Signing the SAML assertion but not the request message would cause a serious security concern: Because there isn't a signature that cryptographically binds the assertion and the request body, the request body could be tampered with during transit. Further, the signed assertion could potentially be hijacked for other unintended uses. Signed or not, an assertion may be hijacked regardless (that is, when there is no message confidentiality), but a signed assertion might give recipients a false sense of security.

7.2 SOAP Message Authentication

Under this architecture, SOAP requests **MUST** be signed if the service provider requires authentication. The digital signature not only provides the ability to verify message integrity, but more importantly serves as the authentication mechanism as well.

The message authentication handler (shown in Figure 7*) determines the authenticity of the message based on the following factors:

- The digital signature for the message can be verified using the designated signature, digest, and canonicalization algorithms;
- The identity of the message sender, represented by the certificate corresponding to the private key that was used to sign the message, can be validated. This involves a series of validation processing achieved jointly by the message handler and the Certificate Validation Service;
- Uniqueness of the message can be determined. Because even a valid, signed SOAP message may be recorded and resent (a replay attack), WS-Security spec states “It is strongly **RECOMMENDED** that messages include digitally signed elements to allow message recipients to detect replays of the message...” (Section 13.2)

After authenticating the message *sender*, the handler creates a new SAML authentication assertion construct to capture this act. Note that this new assertion’s subject contains the sender’s identifier, whereas the assertion contained in the request message, if any, bears the subject of the *originator* (e.g. the end user). Both assertions may serve as relevant inputs to the policy enforcement and policy decision process, as depicted in Figure 10:

*Figure 7 depicts the message authentication handler for a service provider, but the handler also necessary on the service consumer side to authenticate response messages.

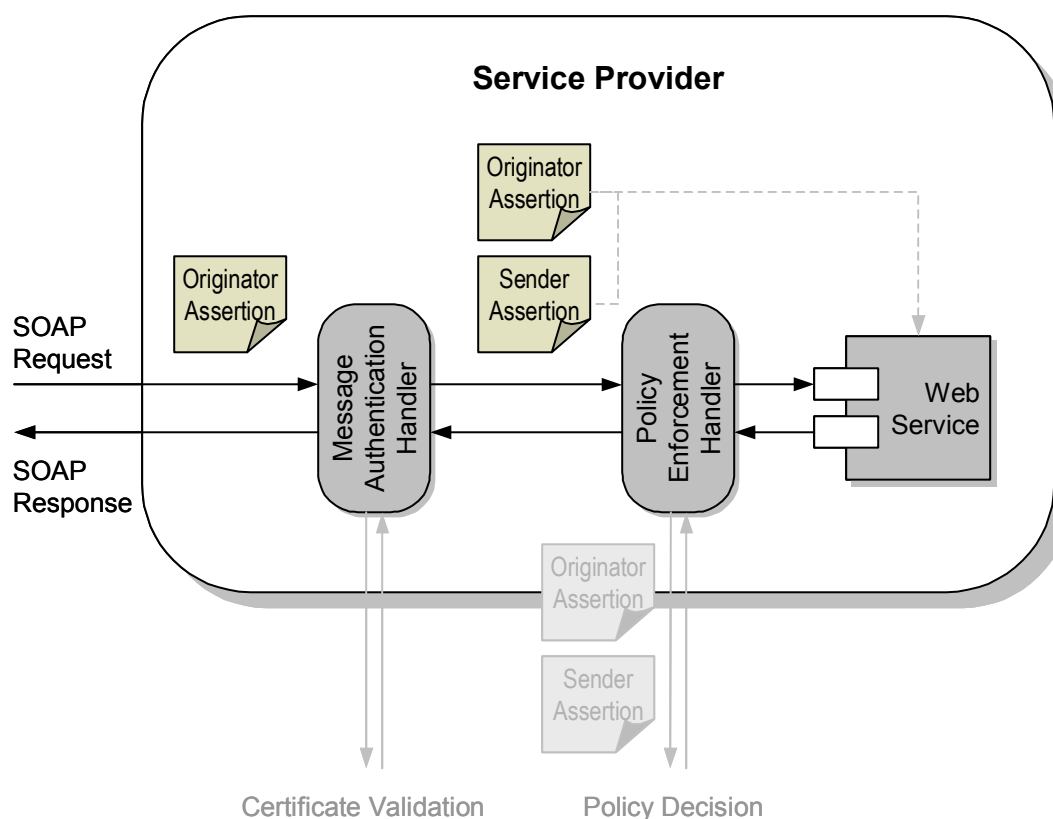


Figure 10 - Message Authentication

In the future, in order to support more sophisticated authorization policies in multi-hop service workflow / collaboration scenarios, the inputs to the policy decision will contain not only the originator assertion and the sender assertion, but the active *intermediaries'* assertions as well, all as part of an "assertion chain".

The diagram also shows that this chain of assertions is made accessible to the Web Service application itself, e.g., via a SOAP message context in JAX-RPC based implementations. This is necessary because the business logic within the Web Service application often may need to rely on user and / or sender identity as well.

Authenticating the message sender is crucial partly because the integrity of the any embedded assertions also depends on it. Fortunately, the WS-Security specification suite along with the XML-DSIG standard clearly defines the message signing syntax and semantics, which have been implemented in many existing commercial or open source toolkits.

In this architecture, asymmetric message signing and verification using DoD PKI certificates is supported. The signed message has roughly the following structure:

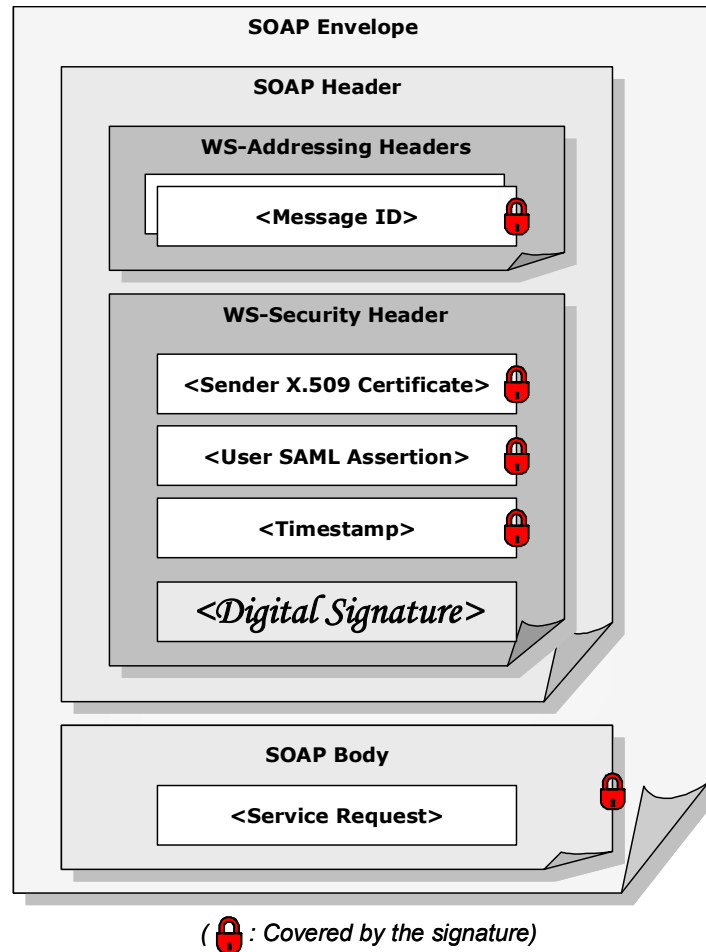


Figure 11 - Signed SOAP Message

As shown in Figure 11, the message sender's signature covers both the header elements (except the signature itself) AND the SOAP body, making sure that none may be tampered or substituted in transit.

This architecture hereby defines the following technical processing rules that implementations need to follow to ensure interoperability among services. These rules are based on the WS-Security core spec, the SAML spec, and the proposed WS-Security SAML Interoperability Scenarios draft document [WSS-SAML-I]. (Note: The following subsections are aimed at a technical audience and may be safely skipped for a high level understanding of the architecture.)

7.2.1 Message Sender Processing Rules

The following rules apply to the message *sender*:

1. The SOAP header MUST contain a WS-Addressing message ID element (i.e. `<wsa:MessageID>`) that contains a globally unique ID (e.g., randomly generated) for this SOAP message. This element is included here to help prevent replay attacks. The message ID element MUST contain the `mustUnderstand="1"` attribute.
2. The Security header MUST contain the `mustUnderstand="1"` attribute.
3. The Security header MUST contain a `<wsse:BinarySecurityToken>` that contains the message sender's X.509v3 certificate*. The `valueType` attribute of the binary token must be `"wsse:X509v3"`.
4. (*For request messages only*) When this message is sent on behalf of another principal, the Security header MUST contain the SAML assertion as defined in section 7.1.
5. The Security header MUST contain a timestamp (i.e. `<wsu:Timestamp>`) as defined in the WS-Security spec. The timestamp MUST contain a `<wsu:Created>` element that records the message creation time relative to the sender's clock. The timestamp MAY also contain a `<wsu:Expires>` element that represents the expiration of the message. As specified in WS-Security SOAP Message Security 1.0 document Section 10, all timestamps MUST be in the UTC format. This architecture further recommends that all timestamps SHOULD have the resolution of milliseconds.
6. The `SignatureMethod` MUST be RSA-SHA1. The `DigestMethod` MUST be SHA-1. The `CanonicalizationMethod` MUST be Exclusive Canonicalization.
7. The signature MUST contain references to the following elements using relative URI:
 - `<wsa:MessageID>`
 - `<wsse:BinarySecurityToken>`
 - `<saml:Assertion>`, if present
 - `<wsu:Timestamp>`
 - `<soap:Body>`
8. The `KeyInfo` element MUST contain a security token reference to the sender's X.509 certificate contained in the `<wsse:BinarySecurityToken>` element.

Please refer to the respective specifications for more syntax and implementation details.

Appendix A.2 contains a sample XML listing of a properly signed SOAP message.

* Note that in the future, this will be relaxed to include any standard referencing mechanisms to the certificate, so that the certificate need not be passed in every message.

7.2.2 Message Recipient Processing Rules

The following rules apply to the message *recipient*:

1. The signed elements defined above **MUST** be verified against the signature using the specified algorithms and transforms and the public key from the sender's certificate.
2. The sender's X.509 certificate **MUST** be validated. The validation may include, but not limited to, (1) it has not expired; (2) its CA chain can be validated against the trusted root; and (3) it has not been revoked based on checking the CRLs. Note that the handler may offload part or all of the certificate validation tasks to an external Certificate Validation Service, which is further discussed in Section 10.2.
3. The handler **MUST** check the Message ID for uniqueness in order to detect replay attacks. The handler **SHOULD** cache the Message ID for a configurable period of time. When message freshness is checked (see below), this period need not be longer than the freshness period.
4. The handler **SHOULD** also check the timestamps for message freshness and discard messages with excessive delays. If the expiration timestamp is present, the handler **SHOULD** discard the message if it has passed the expiration. If the expiration timestamp is not present, the handler **SHOULD** use a "freshness" period instead and discard the message if the message creation timestamp is older than this period. The freshness period **SHOULD** be configurable by the recipient. For example, 5 minutes may be a good default value for interactive scenarios, but it may be considerably longer if asynchronous messaging transport or long network latencies are involved. As stated in the WS-Security spec, "The recipient **MUST** make an assessment of the level of trust to be placed in the requestor's clock." To perform this step, it is **RECOMMENDED** that the machine clocks be synchronized (e.g., using the Network Time Protocol (NTP)).
5. (*For request messages only*) The end user assertion element, if found in the request message, **MUST** be validated against standard SAML processing rules unless the service provider does not rely on it for authorization decisions. In addition, the issuer of the assertion **SHOULD** be checked against an "issuers list" in the trust domain. The implementation of this check may depend on the domain's administrative policies. At the enterprise level, for example, the issuer may be required to be a valid and registered resource in the enterprise UDDI registry.
6. (*For request messages only*) When the message signature is validated, the handler **MUST** create a new SAML authentication assertion for the *message sender*. The assertion **MUST** follow the rules defined in Section 7.1.
7. (*For request messages only*) The message handler **MUST** make the sender's authentication assertion and the end user's assertion (if present) available to the Web Service application, either via a SOAP message context or in an API form.

8. AUTHORIZATION

8.1 Authorization Architecture

Authorization is the means for ensuring that only properly authorized *principals* are able to access *resources* within a system. As defined previously, principals are actors within the system on whose behalf *actions* are taken. Principals can be human (a user) or machine (a system, service, or software process). It is worth noting that systems, services, and software processes may also be resources, and as such can exist as elements on both sides of an authorization request. For example, a service (principal) can request the invocation of another service (resource). Depending on the resource, the types of actions that are possible will also vary. Create, read, update, and delete are typical actions for a data resource. Start, stop, pause, and resume are actions that have no meaning in the context of a data resource, but are quite reasonable for a software process. For Web Services, actions are operations on the service as defined in its WSDL, and are therefore service-specific.

All authorization inquiries have the same general form: Can {Principal X} perform {Action Y} on {Resource Z}? Authorization policy is the means for answering these inquiries. To be more precise, an authorization policy is a set of rules that can be evaluated in response to a request, to arrive at an authorization decision. Policies historically have taken many forms, from access control list (ACL) based to role based; they may be modeled quite differently from one system to another. Policies are an important topic and will be discussed in depth in Section 9.

A solid authorization architecture, however, is concerned not just with policies. Rather, it covers the whole set of components, tools, and data that allows authorization decisions to be made and enforced. The goal of well-defined authorization architecture is to be flexible and extensible enough to accommodate a variety of principals, resources, actions and policies so that a wide range of business use scenarios and stakeholder requirements may be supported.

Figure 12 below is a diagram of a generic authorization architecture that is adopted by this security architecture. Since the primary focus of this architecture is to provide authorization for the invocation of services, the diagram is tailored to that discussion. Therefore, the Policy Enforcement Point (PEP) is depicted between a service Consumer and a service Provider. The general framework, however, is equally valid for other authorization needs. The primary difference would be the location of Policy Enforcement and Policy Decision Points.

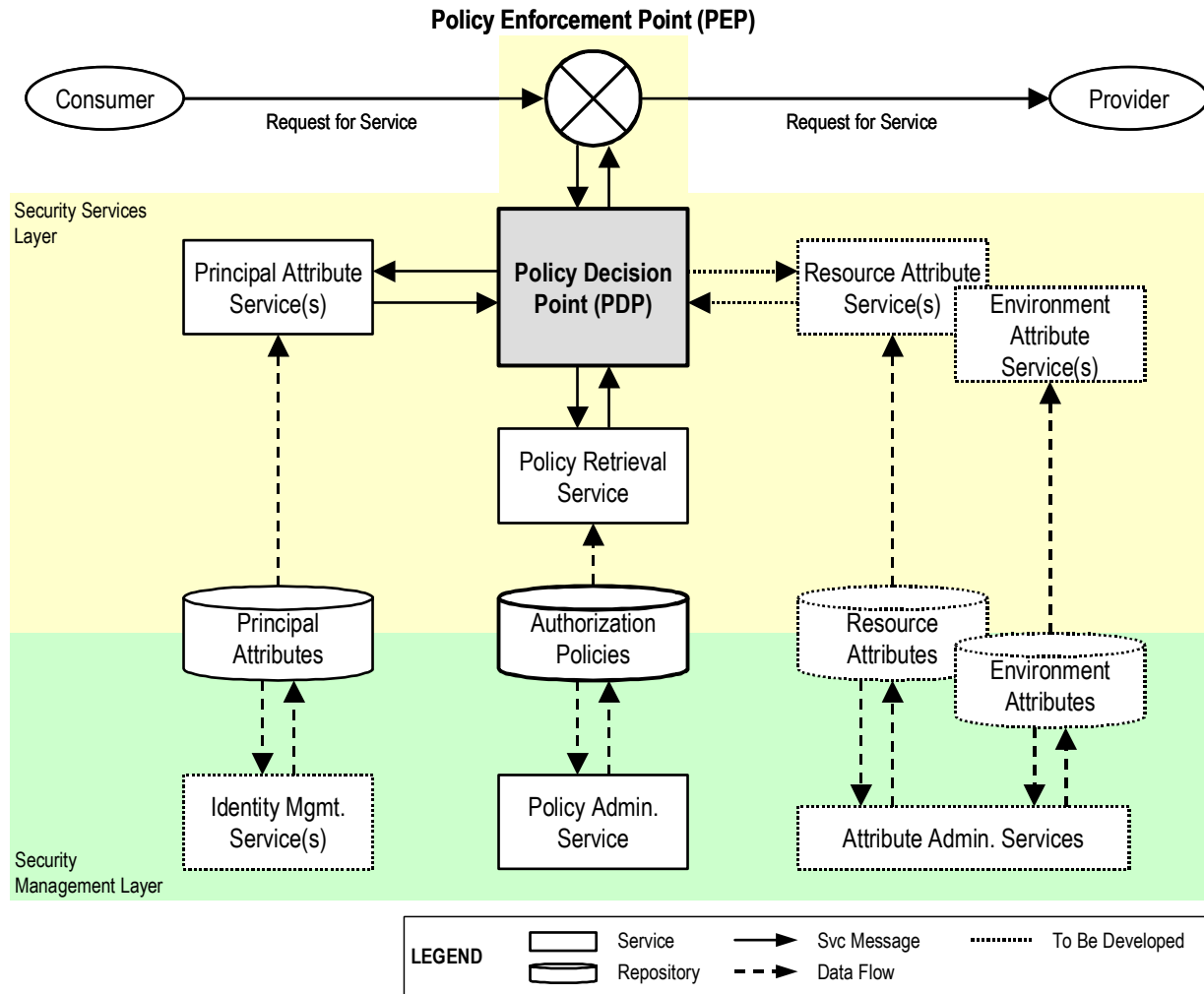


Figure 12 - Authorization Architecture

The PEP is responsible for requesting authorization decisions and enforcing them. In essence, it is the point of presence for access control and must be able to intercept service requests between consumers and providers. For conceptual clarity, the diagram depicts the PEP as a single point. Physically, however, the PEPs would more likely be distributed throughout the system. It is also valid to have more than one PEP on a single message path. For instance, if the PEP was integrated as part of an application service gateway (perhaps within a firewall), it is quite reasonable to encounter more than one gateway, and therefore more than one PEP, between a consumer and a provider. The most important security engineering consideration for the implementation of a PEP is that the system must be designed such that the PEP cannot be bypassed in order to invoke a protected service.

8.2 Two Approaches for Making Policy Decisions

Within the NCES architecture, each provider will incorporate an inbound SOAP message handler that will act as the PEP for all service invocations (see Figure 7 above). As discussed previously, the consumer will digitally sign all service requests such that the handler can verify the integrity of the message and authenticate the identity of the consumer. Once the message is verified, the handler, as the PEP, needs to make an authorization decision request to the PDP. Depending on its implementation preference, the service provider may use two approaches to perform the PDP function:

1. Use an external PDP.

Under this approach, the handler will formulate an Authorization Decision Query message to the Policy Decision Service (PDS), in accordance with the SAML Protocol (SAML-P) 1.1 standard. The handler will place any SAML assertions that were embedded within the header of the original message into the SAML-P Authorization Decision Query as `<evidence>` elements. Now acting as a consumer, the handler will digitally sign the query message with its digital identity before sending it to the PDS, similar to the process described in Section 7.2. Step 3 in Figure 6 illustrates this approach. An actual example of the SAML query (without the digital signature for simplicity) is shown in Appendix A.3.

Authorization decision queries sent to the PDP MUST adhere to the following processing rules:

- The query MUST specify the resource being accessed, using the Web Service's QName as defined in its WSDL;
- The query MUST specify the requested action on the resource, which is the Web Service operation;
- If the request to the service provider was issued on an *originator's* behalf*, the query's `<subject>` element MUST carry the *originator's* identifier, and the *originator's* assertion MUST be added as the supporting `<evidence>`[†];
- Otherwise, the query's `<subject>` element MUST carry the (immediate) *sender's* identifier, and the assertion created by the message authentication handler for the service consumer MUST be added as `<evidence>`.

Note that in this release, the policy decisions are based on one and only one principal. That is, if the query is on an *originator's* behalf that is different from the

* The originator may or may not be the end user, see Section 6.2

† The query MAY also contain additional attributes regarding principals. This alternative allows for the possibility that principal attributes may be obtained prior to making an authorization decision query.

immediate sender, the sender principal is ignored in the decision process. *In the future, the decision may be further based on the principals of the sender as well as all other intermediaries involved in the invocation chain* (see the Future Work section for more discussions on this topic).

Upon receiving the authorization decision query, the PDS will parse it to determine the resource and action being requested. It will use this information to retrieve the appropriate policy from the Policy Retrieval Service (*note: depending on whether or not the policy services are collocated, PDS may choose to directly retrieve policies from the policy store; the actual choice is up to the implementer*).

Additionally, the PDS may need to retrieve additional principal, resource, and/or environment attributes in order to evaluate the policy. As shown in Figure 6, the attributes may be retrieved from the respective Attribute Services.

After the policy is evaluated, the result is rendered as either *Permit*, *Deny*, or *Indeterminate*, in the form of a SAML Authorization Decision Assertion. The assertion is returned to the requesting PEP using a standard SAML Protocol response message. This is shown as Step 4 in Figure 6 with an XML example in Appendix A.4.

2. Perform the PDP function locally.

Alternatively, the service Provider could choose to deploy a local PDP. In this case the PEP and PDP are essentially combined within a message handler, making the SAML query to the PDS unnecessary, but the policy evaluation process remains the same.

To enforce enterprise policies, the PEP+PDP message handler will need to retrieve the appropriate policy from the Policy Retrieval Service, as well as necessary principal attributes from the Principal Attribute Service, just like the way PDS does it. The policy is returned using the standard XACML syntax. This is shown as Step 3-alt. in Figure 6 and an example of a XACML policy is listed in Appendix A.5.

The two approaches both have advantages and disadvantages. The external PDP relieves the service provider from the complexities of policy evaluation and policy retrieval, but an internal PDP may offer better performance and availability due to tight integration of PEP and PDP. The actual choice between an external vs. local PDP depends on the service provider as well as the trust domain policies it is required to satisfy; in fact the trust domain may have a mix of both approaches among different providers.

Regardless of the two approaches used, a service provider may need to implement additional security checks by itself. For example, a data service may need to filter a large number of data objects based on the consumer's privileges before returning them in the SOAP response. This would be too inefficient for the PEP to do, as it involves examining the SOAP message payloads. In this scenario, the Web Service retrieves relevant principal attributes from the Principal Attribute Service as necessary, queries the Policy Retrieval Service to retrieve the relevant policy rules (in XACML syntax), and then applies the policies on top of the service level policy enforcement the message handler.

8.3 Policy Decision Implementation Considerations

No matter where it is located, the PDP may find it necessary to obtain more attributes to perform the policy evaluation and render a decision. Three possible processing options are described below:

1. The PDS could immediately retrieve all attributes that are available from each of the attribute services prior to evaluating the policy in any fashion. Once all attributes have been retrieved, the policy is evaluated and the decision is rendered as *Permit*, *Deny*, or *Indeterminate*.
2. The PDS could parse the policy inputs and determine whether all of the necessary attributes have been obtained. If not, the PDS could then initiate attribute requests to the appropriate attribute services to retrieve the needed information. Once all necessary attributes have been retrieved, the policy is evaluated and the decision is rendered as *Permit*, *Deny*, or *Indeterminate*. If any of the necessary attributes could not be retrieved, the decision is rendered as *Indeterminate* immediately.
3. The PDS could attempt to evaluate the policy with the information it already has. If a *Permit* or *Deny* outcome is reached, the decision has been rendered. If the outcome is *Indeterminate*, the PDS would then initiate attribute requests to the appropriate attribute services – as specified in option 2.

The choice of options will not impact the ultimate result of the policy evaluation. The choice is fundamentally an implementation decision that should be made based on performance optimization. The optimal choice will depend on the profile of authorization policies within a given system environment. One important consideration to be made is whether the PDS will support the *Indeterminate* decision result. The *Indeterminate* result is defined in the SAML standard as a method for a SAML authorization authority to positively affirm that it is unable to render a decision. Some reasons for why this result could occur include, a) the PDS cannot locate a policy for the specified resource within the domain's Authorization Policy repository, b) one or more attribute values needed to evaluate the policy could not be retrieved, c) the PEP that sent the query is not recognized as a PEP within the domain, etc. Regardless of the

reason, the PDS MAY be configured to provide a *Deny* result in these cases. In this way, the PDS can remove any ambiguity on the part of the PEP for how to enforce the decision.

Attribute Services are supporting services to the PDS. As discussed they provide information relative to Principals, Resources, and the trust domain Environment. As the name implies, Principal attributes are bound to a principal identity and could include values such as identity, aliases, email address, roles, communities of interest, clearance, formal access approvals, citizenship, organizational designator, office, phone number, etc. Resource attributes are those attributes associated with resource objects and could include a wide variety of metadata values such as resource identifier, type, provider, security label, hash or checksum, keywords, etc. Environment attributes are those attributes that are not associated with a principal or resource, but are still useful as part of an authorization policy. Examples of environment attributes could include values such as maximum security level, hours of operation, current time, geographic location, time zone, etc. These values may be common across a trust domain and therefore not treated as resource attributes.

Attribute query messages will be formulated in accordance with the SAML-P spec. Likewise, each attribute service will respond in the form of a SAML attribute assertion contained in a SAML Protocol response.

It is important to note that, attributes that are relevant to the policy decision process are sometimes sensitive information that needs to be protected. For example, it is critical to prevent principal attributes from being harvested by a rogue PDP for malicious purposes (e.g. data mining on users). Future versions of this architecture will address this issue.

9. AUTHORIZATION POLICIES

In the previous section, a *normative* authorization architecture was defined, with clearly defined component boundaries and standards-based message exchange mechanisms for policies and policy decisions. However, as mentioned earlier, the underlying authorization policies, managed by the Policy Administration Service, could take many forms and may differ from domain to domain or from one resource type to another. For this reason, it is the intention of this document to keep the policy management aspect of the authorization architecture *non-normative*, allowing domain-specific access control models to be plugged-in. In this section, a basic Role Based Access Control (RBAC) model is described, that SHOULD be adopted for authorizing Web Service invocations. In the future, this model may evolve to a broader attribute-based model that can accommodate policy decisions based on many attributes including roles.

9.1 The RBAC Model

To reiterate, authorization policy is the means for answering the query “Can {Principal X} perform {Action Y} on {Resource Z}?” An access control list or ACL is a simple form of authorization policy found in most file management systems today. An ACL, for example, might specify that an explicit list of users may read a particular file. This is an example of identity based access control (IBAC). But access lists are difficult to maintain. Thus, other approaches have been devised such as role based access control (RBAC), where the identity of individual users is replaced in authorization policies by a role. In this way, roles can be assigned or unassigned to users without needing to modify the access policies themselves.

The RBAC model has the following benefits:

- Compared with IBAC, RBAC adds levels of indirection between identities and resources. Because permissions no longer need to be repeatedly assigned to individual users, RBAC scales much better and significantly reduces administration overheads.
- RBAC is highly flexible and can have many variants with sophisticated features (e.g. role hierarchies, dynamic separate of duty constraints), making it possible to meet more complex business needs.
- Compared with Mandatory Access Control (MAC) policies that are based on coarse labeling of subjects and objects, RBAC provides more granularity of assigning permissions, so that users are not granted (and hence use) more access than they need.
- RBAC maps intuitively to the way business roles and responsibilities are managed, and is therefore easy to understand and use.

Because of these benefits, RBAC has become the prevalent access control model today and widely deployed a wide variety of systems and products. Its inherent flexibility, however, also resulted in the lack of general agreement on its definition and features, which has created uncertainty and confusion of its usage and meaning. Recently, a voluntary RBAC standard has been proposed by the National Institute of Standards and Technologies (NIST) to resolve this situation. The standard defines a “reference model” that formalizes RBAC features, and then describes the functional specifications of those features [RBAC].

For access control of Web Service invocations, we hereby describe a simple RBAC model that is consistent with the Core RBAC Reference Model defined in the NIST standard. The model is shown in Figure 13 below.

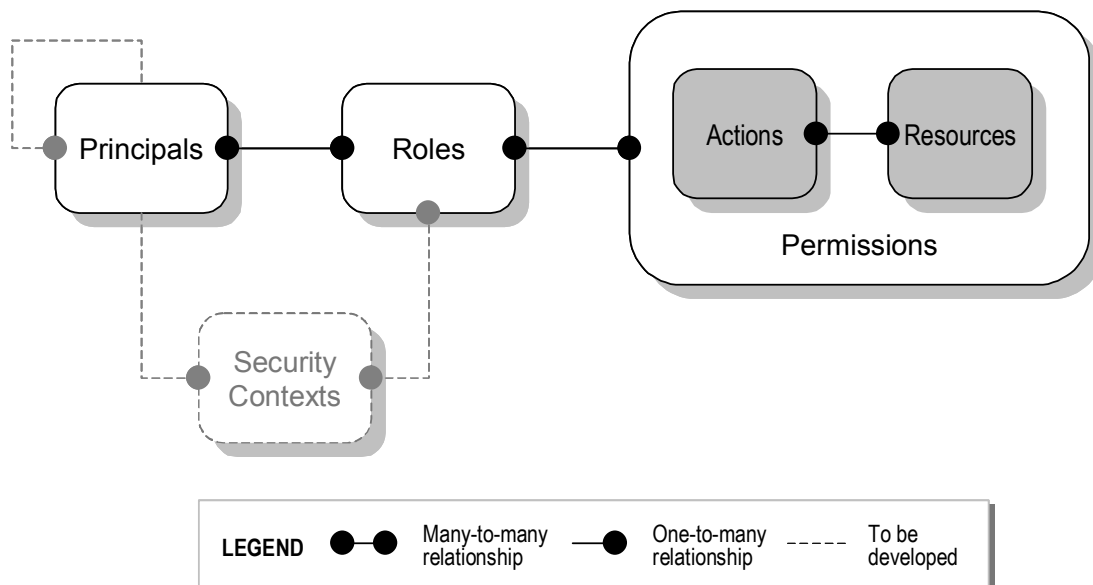


Figure 13 - Basic RBAC Model

Technically, roles in this diagram are a means for many-to-many relationship between principals and permissions. However, it is important to note that roles should not be viewed simply as a technical mechanism for easier grouping of users or permissions. They should map to business meanings and responsibilities. As defined in the NIST standard, a role should be “a job function within the context of an organization with some associated semantics regarding the authority and responsibility conferred on the user.” In the NCES context, depending on whether the trust domain is an enterprise (or COI, or local) domain, enterprise (or COI, or local) roles SHOULD be clearly defined and centrally managed by the domain.

Permissions in the above model represent approval to perform an action on one or more resources. Note that a resource (e.g. a Web Service) usually has multiple actions (e.g.

service operations defined in WSDL), while the same action may be applicable to multiple resources. For example, the “getAttribute” operation may be defined for Principal, Resource, and Environment Attribute Services.

In the near future two additional features may be added to this basic model:

1. *Principal hierarchies*. In a large scale business environment such as the DoD enterprise, there could be millions of users and other types of principals. For ease of administrative duties hierarchical groups of principals MAY be defined so that the number of principals assigned to a role doesn’t get out of control. For example, if 50 contractors from company A and 50 contractors from company B need to be assigned the “Demo User” role, adding 100 principals in the role membership list could be a very tedious job. It would be more convenient and more manageable to create two groups, “Company A contractors” and “Company B Contractors”, and then add the two groups under the role membership. When doing so, however, it is imperative to maintain the clear distinction between principal groups and roles. Principal groups are purely convenience constructs and should not have authorities and responsibilities directly attached to them.
2. *Context activated roles*, also known as “session roles” in the NIST standard. Sessions are common in an enterprise environment. For example, an end-user logging in to a portal server from a thin client usually has a session or multiple sessions that last for a certain period of time. In the NCES architecture, sessions are generalized into a broader concept of *security contexts*, which not only exists in edge applications but spans service providers as well. Unlike user sessions which are primarily authentication oriented (so that users do not have to log in for every HTTP request), security contexts also have important authorization uses. In particular, it is sometimes desirable to allow only a subset of a user’s roles to exist under a certain context. For example, when a DoD employee works in the office, all his or her roles are activated to access the authorized business functions. When logging in from home, however, the employee may still be able to read news on the web portal, but the “Procurement Manager” role will not be activated to ensure that sensitive information cannot be accessed from unprotected computers. As shown in Figure 14, a principal may be associated with a set of security contexts and a security context may be associated with a subset of activated user roles. This feature provides further expressive power to the RBAC model.

Under this model, authorization policies are naturally represented as rules that assign permissions to roles, as shown in Figure 14 below:

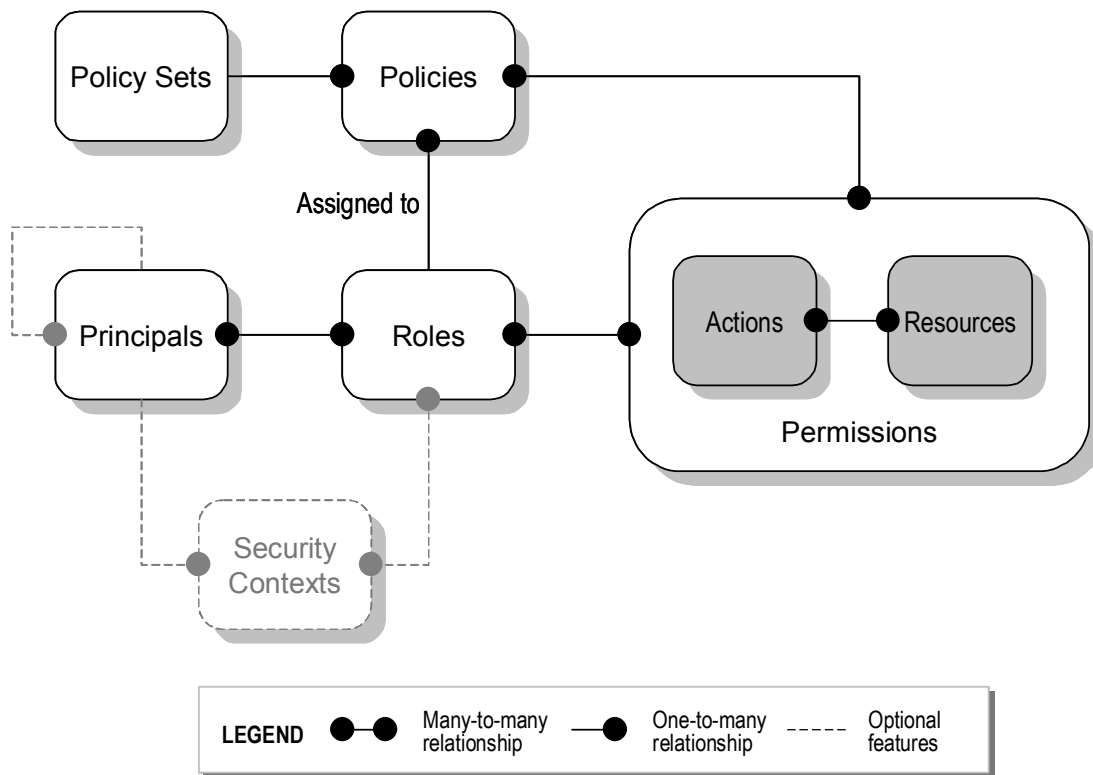


Figure 14 - RBAC Based Policies

As shown in the diagram, policies MUST be associated with roles, not directly with principals. Generally, for clarity and granularity sake, a policy SHOULD be assigned to one role. The policy MAY contain permissions for multiple resources, though. The actual XACML exchange format of the policy may vary depending on the actual query. For instance, if a XACML query is concerned with a specific action on a specific resource, then only that part of the policy concerning the requested action may be returned.

9.2 Looking Ahead: the Attribute Based Approach

Through the use of multiple roles, policies can be formed that satisfy more complex business needs. However, for even more complex needs, roles alone are inadequate to provide the desired flexibility. Within the federal government, particularly within the DoD and Intelligence communities, access control must also be based on the sensitivity of information. Whereas IBAC and RBAC are primarily discretionary access control (DAC) mechanisms, they are not best suited to address these mandatory access control (MAC) requirements.

The goal of a well-engineered authorization framework is to be flexible enough to accommodate a variety of principals, resources, actions and policies such that it is extensible to a wide range of business use scenarios. In response to more complicated

business needs, authorization mechanisms have been migrating, slowly but inexorably, towards a more flexible *attribute based* approach. Since identities and roles can be viewed as nothing more than attributes of principals, both IBAC and RBAC can be wholly absorbed into an attribute-based mechanism. Attribute-based authorization policies have some distinct advantages over other approaches. First, an attribute-based approach recognizes from its inception that a flexible access control policy cannot be locked into evaluating only one dimension of a principal (such as an identity or role). For example, in order to provide proper controls for accessing classified information it is necessary to consider various other principal attributes such as clearance level, formal access approvals, or citizenship. Second, an attribute-based approach takes into consideration that there are other attributes that are relevant to authorization policies besides those associated with a principal, such as resource or environment attributes.

The attribute-based approach will be explored in more detail in future versions of this document.

10. OTHER TOPICS

10.1 Message Confidentiality

When required to counter identified threats, two general options are available to achieve confidentiality for service messages:

- Rely on underlying communications security: Unencrypted SOAP messages may be acceptable if the underlying communications infrastructure provides adequate confidentiality to protect transmissions. Confidentiality can be provided at several layers of the communications stack; transport layer (e.g., SSL / TLS), network layer (e.g., IPSEC, inline network encryptors), and physical / link layer (e.g., link encryptors).
- Encrypted SOAP messages: Message encryption is recommended when transmission confidentiality is not provided. When XML-Encryption is used, the WS-Security processing logic within the SDK message handlers may also be responsible for encrypting and decrypting the SOAP message.

These options are not mutually exclusive and therefore may be employed together as part of a defense-in-depth strategy. Using at least one of these options is recommended. Due to performance concerns involved in message level encryption, the transmission confidentiality techniques may be preferable in most situations in the near term. If message encryption is not employed, XML encryption may still be employed at the implementer's discretion to provide confidentiality for elements within the body of a message (data confidentiality).

10.2 Use of DoD PKI

As described earlier, the Credential Management Services group of services provides the primary interface to the DoD PKI. The DoD PKI consists of the products and services that provide and manage X.509 certificates for public-key cryptography. Specific services provided by the DoD PKI include:

- Key generation, storage, and recovery (encrypt / decrypt keys)
- Certificate generation, update, renewal, rekey, and distribution
- Certificate Revocation List (CRL) generation and distribution
- Directory management of certificate related items
- Certificate token initialization, programming, and management (NIPRnet)

Since the Certification Validation Service (CVS) is the only service offered in the current architecture that is related to credential management, a few CVS related implementation issues are discussed below.

10.2.1 Revocation Status Checking vs. Full Certificate Validation

As mentioned earlier in Section 4.5, a client (message handler) may delegate part or all of the certificate validation tasks to the CVS. Two reasonable alternatives are: (1) use the CVS to perform certificate revocation status checking, in which case the CVS functions similar to an OCSP responder, or (2) use the CVS to perform the entire certificate path validation. Both alternatives provide significant benefit to the message handler. By offloading revocation status checking to the CVS, the handlers do not need to download, store, maintain, and process certificate revocation lists (CRLs) for the entire PKI. Distribution of CRLs within an enterprise can be a very difficult and costly process. Within the DoD PKI, this process is further complicated by the extremely large size of many DoD CRLs.

When the CVS performs the entire certificate validation process, the message handler is only responsible for verifying the digital signature of the message. The handler is relieved of the complexities of X.509 processing (e.g., certificate path construction, name subordination checking, full certificate parsing, certificate extension processing, as well as revocation status checking). This handler is also relieved of maintaining or locating many (or perhaps all – see 10.2.2) of the CA certificates and CRLs that are needed to build and validate certificate paths. In either case, the message handler (as the relying party) must be able to authenticate the CVS responses so that it can verify that they were sent from a responder that it trusts. This is most easily satisfied using signed responses, although not without some complications (see 10.2.2).

Under the current architecture, service providers and consumers are REQUIRED to perform certificate path validation, but MAY rely on the CVS to provide certificate revocation status checking. Requests to and responses from the CVS MUST be in the form of a signed SOAP message as discussed in Section 7.2.

The use of CVS for full certificate path validation is not currently supported, but remains an option under consideration for future releases.

10.2.2 CVS Certificate Options

In both cases described in the previous section, the CVS serves as a trust responder, whereby a message handler delegates some portion of its trust processing to the CVS, which it authenticates through a signed response. As such, the benefits are not gained without some trade-off of risk. More specifically, the message handler faces two options when authenticating the CVS response: it either chains the CVS certificate to a Root certificate, or explicitly trusts that certificate.

1. Chaining to the Root

Without a trust responder, all end-entity certificates and CRLs are signed by either the Root CA or a subordinate CA issued from the Root. Therefore, a message handler can base its authentication on the ability to chain back to a Root certificate that it explicitly accepts as a trust anchor. With a trust responder it is possible to maintain this single trust anchor by using a certificate issued from within the PKI. However, in order to authenticate responses, the message handler would still need to perform certificate path validation including revocation checking for the responder. As a result, the message handler would need to maintain certificates for the Root CA and the subordinate CA that issued the responder's certificate. It would also need to maintain the CRLs issued by the Root and subordinate CAs. To reduce this need, it is possible that trust responder certificates could be issued directly from the Root CA. This would be optimal from the standpoint of the message handler because responder certificates would chain directly to the Root, and only the Root CRL would be needed. However, this approach is not desirable from the standpoint of the DoD PKI for various policy and operational reasons. Moreover, since there is a potential need for many trust responders to exist across the enterprise, it would be better to issue these certificates from subordinate CAs. It is important to note that using an existing DoD CA is also not ideal due to the potentially large size of the CRL.

Therefore, in order to maintain the Root CA as the single trust anchor, the best overall alternative would be to establish a subordinate CA under the Root dedicated to issuing only trust responder certificates – since this would allow the CRL for this CA to remain small (relative to the other DoD subordinate CAs.)

2. Explicit trust of responders

One way to maintain the full benefits of trust responders is to have each message handler explicitly trust the responder's certificate as a trust anchor – similar to the way a Root certificate is trusted. In this way, the message handler can directly verify response message signatures, and does not need to perform any certificate path validation or revocation status checking for the responder itself. The only operational complexity this introduces is the need to distribute and trust the responder certificates to each message handler, which makes this option very attractive. However, explicit trust is not without its drawbacks. This approach introduces additional risk to the solution and therefore lowers overall security. The risk that is introduced relates to the potential compromise of the responder's private key.

Without explicit trust, the only trust anchor is the Root CA. The probability of compromising the DoD Root private key is virtually nil, since the Root CA is a standalone machine, and there are an extraordinary number of technical, physical, and procedural controls in place to prevent such a compromise. A trust responder,

by its very nature, must be connected to the network and up and running at all times. Therefore, probability of compromising the responder's private key is substantially higher because the responder platform is susceptible to network attack. The impact of a responder compromise depends on the nature of the responder. Compromise of a revocation status responder allows an adversary to forge responses. The two potential results are a) the responder can potentially cause others to trust certificates that have been revoked by the PKI, or b) the responder can potentially cause all certificates to appear to be revoked, thus creating a denial of service. Compromise of a full certificate path validation responder is far more serious. Since the message handler delegates all validation to the responder, an adversary can cause any certificate to be trusted, regardless of its true origin or validity. This is functionally equivalent to a Root key compromise for any message handler that trusts that responder.

One way to mitigate this risk is to employ a hardware cryptographic module at the responder (e.g., nCipher, Chrysalis) to provide strong protection for the private key. Another mitigation could be to limit the validity period of responder certificates to a reasonable period of time (e.g., 30 days). However, this would introduce the operational burden of periodically redistributing the certificate.

There is no absolute requirement for the responder certificate to be issued from the PKI when using explicit trust. Since the certificate must be distributed as a trust anchor it is possible to use a self-signed certificate. However, using a certificate issued from the PKI is preferable since the distribution can be conducted in-band. Since there is no alternate means for validation of a self-signed certificate by implementers prior to installation as the trust anchor, it must be distributed via an out-of-band process. The only compelling reason to use a self-signed certificate would be if an appropriate certificate could not be obtained directly from a DoD PKI CA due to operational or policy constraints.

Due to the complexity and operational requirements involved in setting up special subordinate CAs for CVS, we leave the second option for further discussions.

In this version of the architecture, service consumers and providers are **REQUIRED** to explicitly trust the CVS' certificate. They **MUST** validate the signed CVS responses using the pre-distributed CVS certificate.

10.3 Beyond Trust Domain Boundaries

A new set of security challenges arise when a service consumer attempts to invoke a service in a different trust domain, largely because the consumer and provider are governed by different authentication and authorization policies. Issues to consider may include the following:

- 1849
- 1850 ➤ Trust domain federation. A trust relationship, governed by mutually agreed-
- 1851 upon policies, must be established between the two domains before resource
- 1852 access can occur. The relationship is one-way by nature, but two domains may
- 1853 have mutual trust by establishing two one-way relationships;
- 1854 ➤ Discovery of services in foreign domains. Service registries need to be aware of
- 1855 internal vs. external discoveries and exert different access controls on service
- 1856 visibility;
- 1857 ➤ Mechanisms for exchanging user attributes. For example, the source domain
- 1858 may not be willing to release all user attributes to the target domain, and may
- 1859 choose to attach some “public” attributes along with the authentication assertion;
- 1860 ➤ Federation of identities, when a principal has different identities and / or
- 1861 credentials in the two domains;
- 1862 ➤ MLS concerns, when the two domains are not at the same security level;
- 1863 ➤ PKI interoperability concerns, esp. if the two domains have different CA roots
- 1864

1865 These topics are beyond the scope of this release and will be covered in future versions

1866 of the architecture.

1867

1868

11. FUTURE WORK

The following are identified as items that need to be addressed in future iterations of the architecture:

1. Lack of target destination in a Web Service request.

Because the identifier of the target service is not part of the SOAP protocol (SOAP lets underlying transport protocols such as HTTP to handle this), a Web Service request may potentially be hijacked and replayed to some other service provider that happens to support the same WSDL operation. A simple solution is to include a “target service URI” in the SOAP header, which is covered by the message signature. Currently emerging standards such as WS-Addressing [WSADDR] are addressing this problem; they be evaluated for use in this architecture in the future.

2. Relay of trust in service chaining.

As described in Section 7 and 8, a service provider’s authorization decision for an incoming service request is built on both the immediate sender of the message *and* the assertion of the end user on whose behalf the request is made. In the service chaining scenario, shown in Figure 8, assuming the requests are on the end user’s behalf, the sender of the message are no longer the issuer of the end user’s authentication assertion for the second and all following requests. This implies that a service provider will not accept the incoming request unless *all* intermediaries along the invocation path are trustworthy – that they haven’t intentionally tampered with and / or misused the original assertion. One potential risk lies in the possibility for an intermediary to hijack a valid user assertion and place that in malicious messages. This risk is not mitigated even if the original assertion is signed by the portal, because the downstream service provider still have to trust that the *bindings* between the assertion and the request bodies haven’t been tampered with by any of the intermediaries.

This problem cannot be addressed by putting a target service URI in the original SOAP request, either, because in a dynamic service collaboration environment the final target service(s) may not be known when the original request is made.

A possible solution is to include an “intended use” attribute in the original authentication assertion, which defines the boundary of this unit of work or business transaction, so that if the assertion is misplaced and use for some other purposes, the recipient of the message is able to detect and reject it. This approach needs to be further researched and defined.

3. Richer policy decision semantics.

1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952

Future work in this area may include:

- Attribute based policy models;
- Making policy decisions based on not just the end user identity, but also on the principals of intermediaries in the invocation path;
- Introducing resource and environment attributes as policy constraints;
- Context activated roles, as mentioned in Section 9.1;
- Introducing role hierarchies in the basic RBAC model;
- Introducing static or dynamic Separation of Duty constraints into the basic RBAC model

4. Security context establishment.

There are some proposed standards that may address this issue, such as WS-Trust, WS-SecureConversation, and WS-Federation.

5. Cross-trust domain service based collaboration.

This is a challenging problem area with a lot of topics, including identity federation, cross-domain authentication and authorization, MLS support, etc.

6. Content or data level security.

That is, how we can better protect sensitive data in a SOA environment. This is of particular relevance to content discovery services.

7. Integration with other net-centric enterprise services.

Here we need to address the integration of the Security Services with Service Discovery, Enterprise Service Management (ESM), Messaging Services, and others.

8. Definition of enterprise level security attributes and taxonomies.

Before we can exchange identity, resource, and environment attributes among service providers (including NSS providers), we need to identify and define those attributes. This is also important to cross-domain security.

9. Definition of enterprise auditing and logging services.

They are essential for accountability in a brokered trust environment, esp. in the absence of positive mandatory access controls.

1953 **10. Protection of sensitive attributes.**

1954

1955 Principal and resource attributes need to be protected so that they cannot be
1956 harvested for data mining or other malicious purposes.

1957

1958 **11. Support for “thick” Clients.**

1959

1960 In addition to end users accessing enterprise services via a web portal from a web
1961 browser (“thin” client), there are other applications that are not web-based (e.g. Java
1962 desktop applications or legacy systems), but may also need to access the services.
1963 Authentication and authorization of such SOAP requests will be addressed later.

1964

1965

1965 A. MESSAGE EXAMPLES

1966 A.1 SAML Assertion Element Created by Portal

```

1967 <?xml version="1.0" encoding="UTF-8"?>
1968 <saml:Assertion
1969   xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
1970   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
1971   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1972   xsi:schemaLocation="urn:oasis:names:tc:SAML:1.0:assertion ..."
1973   MajorVersion="1"
1974   MinorVersion="1"
1975   AssertionID="A4061B4E-61E9-200F-6115-209A56B8E384"
1976   Issuer="mydomain\my_portal"
1977   IssueInstant="2004-01-27T06:00:10Z">
1978   <saml:AuthenticationStatement
1979     AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:X509-PKI"
1980     AuthenticationInstant="2004-01-27T06:00:00Z">
1981     <saml:Subject>
1982       <saml:NameIdentifier
1983         Format="urn:oasis:names:tc:SAML:1.1:nameid-
1984         format:X509SubjectName">
1985         CN=John Doe,OU=NCES,DC=DISA,DC=mil
1986       </saml:NameIdentifier>
1987     </saml:Subject>
1988   </saml:AuthenticationStatement>
1989 </saml:Assertion>
1990
1991
1992

```

1992 A.2 Signed SOAP Request

```

1993 <SOAP-ENV:Envelope
1994   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
1995   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
1996   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1997   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
1998   xmlns:m0="http://echo.example.org/schema"
1999   xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility/"
2000   xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing/">
2001   <SOAP-ENV:Header>
2002     <!-- WS-Addressing Header -->
2003     <wsa:MessageID wsu:Id="msgid">urn:A4061B4E-61E9-200F-6115-
2004     209A56B8E384</wsa:MessageID>
2005     <!-- WS-Security Header -->
2006     <wsse:Security
2007       xmlns:wsse="http://.../secext"
2008       soapenv:mustUnderstand="true">
2009       <wsse:BinarySecurityToken
2010         wsu:Id="binarytoken"
2011         ValueType="wsse:X509v3"
2012         EncodingType="wsse:Base64Binary">
2013         MIIIEZzCCA9CgAwIBAgIQEmtJZc0...
2014       </wsse:BinarySecurityToken>
2015     <saml:Assertion
2016       wsu:Id="samlassertion"

```

```

2017      xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
2018      xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
2019      xmlns:xsi="http://.../XMLSchema-instance"
2020      xsi:schemaLocation="urn:oasis:names:tc:SAML:... .."
2021      MajorVersion="1"
2022      MinorVersion="1"
2023      AssertionID="A4061B4E-61E9-200F-6115-209A56B8E384"
2024      Issuer="example_domain\my_portal"
2025      IssueInstant="2004-01-27T06:00:10Z">
2026          <saml:AuthenticationStatement
2027              AuthenticationMethod="...:X509-PKI"
2028              AuthenticationInstant="2004-01-27T06:00:00">
2029              <saml:Subject>
2030                  <saml:NameIdentifier
2031                      Format="...:X509SubjectName">
2032                      CN=John Doe,OU=NCES,DC=DISA,DC=mil
2033                  </saml:NameIdentifier>
2034              </saml:Subject>
2035          </saml:AuthenticationStatement>
2036      </saml:Assertion>
2037      <wsu:Timestamp wsu:Id="timestamp">
2038          <wsu:Created>2004-01-27T06:00:30Z</wsu:Created>
2039      </wsu:Timestamp>
2040      <!-- Digital Signature -->
2041      <ds:Signature xmlns:ds="http://www.w3.org.../xmldsig#">
2042          <ds:SignedInfo>
2043              <ds:CanonicalizationMethod
2044                  Algorithm=".../REC-xml-c14n-20010315"/>
2045              <ds:SignatureMethod
2046                  Algorithm=".../xmldsig#rsa-sha1"/>
2047              <ds:Reference URI="#msgid">
2048                  ...
2049              </ds:Reference>
2050              <ds:Reference URI="#binarytoken">
2051                  ...
2052              </ds:Reference>
2053              <ds:Reference URI="#samlassertion">
2054                  ...
2055              </ds:Reference>
2056              <ds:Reference URI="#timestamp">
2057                  ...
2058              </ds:Reference>
2059              <ds:Reference URI="#msgbody">
2060                  ...
2061              </ds:Reference>
2062          </ds:SignedInfo>
2063          <ds:SignatureValue>
2064              CwP3qte8VosbgUnQnF+...
2065          </ds:SignatureValue>
2066          <ds:KeyInfo>
2067              <wsse:SecurityTokenReference>
2068                  <wsse:Reference URI="#binarytoken"/>
2069              </wsse:SecurityTokenReference>
2070          </ds:KeyInfo>
2071      </ds:Signature>
2072      </wsse:Security>
2073      </SOAP-ENV:Header>

```

```

2074      <SOAP-ENV:Body wsu:Id="msgbody">
2075          <m:echo xmlns:m="http://echo.example.org">
2076              <m0:EchoString>Hello World</m0:EchoString>
2077              <m0:NumEchoes>1</m0:NumEchoes>
2078          </m:echo>
2079      </SOAP-ENV:Body>
2080  </SOAP-ENV:Envelope>

```

A.3 SAML-P Authorization Decision Query

```

2083  <?xml version="1.0" encoding="UTF-8"?>
2084  <soapenv:Envelope
2085      xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
2086      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
2087      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
2088      <soapenv:Body>
2089          <samlp:Request
2090              IssueInstant="2004-01-31T13:00:58.287Z"
2091              MajorVersion="1" MinorVersion="1"
2092              RequestID="ID_2d10e285-42d4-4926-984e-fab8ea72d32a"
2093              xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol">
2094              <samlp:AuthorizationDecisionQuery
2095                  Resource="urn:mydomain:HelloWorldService">
2096                  <saml:Subject
2097                      xmlns:saml="urn:...:SAML:1.0:assertion">
2098                      <saml:NameIdentifier
2099                          Format="...:X509SubjectName"
2100                          NameQualifier="...">
2101                          CN=John Doe,OU=NCES,DC=DISA,DC=mil
2102                      </saml:NameIdentifier>
2103                  </saml:Subject>
2104                  <saml:Action
2105                      xmlns:saml="urn:...:SAML:1.0:assertion">
2106                      helloWorld
2107                  </saml:Action>
2108                  <saml:Evidence
2109                      xmlns:saml="urn:...:SAML:1.0:assertion">
2110                      <saml:Assertion
2111                          AssertionID="..."
2112                          IssueInstant="..."
2113                          Issuer="..."
2114                          MajorVersion="1" MinorVersion="1">
2115                          <saml:Conditions
2116                              NotBefore="..."
2117                              NotOnOrAfter="...">
2118                          <saml:AuthenticationStatement
2119                              AuthenticationInstant="..."
2120                              AuthenticationMethod="urn:
2121                              oasis:names:tc:SAML:1.0:am:password">
2122                              <saml:Subject>
2123                                  <saml:NameIdentifier
2124                                      Format="..."
2125                                      NameQualifier="...">
2126                                      CN=John Doe, ...
2127                                  </saml:NameIdentifier>
2128                              </saml:Subject>

```

```

2129         </saml:AuthenticationStatement>
2130     </saml:Assertion>
2131 </saml:Evidence>
2132 </samlp:AuthorizationDecisionQuery>
2133 </samlp:Request>
2134 </soapenv:Body>
2135 </soapenv:Envelope>

```

A.4 SAML-P Authorization Decision Response

```

2138 <?xml version="1.0" encoding="UTF-8"?>
2139 <soapenv:Envelope
2140     xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
2141     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
2142     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
2143     <soapenv:Body>
2144         <samlp:Response
2145             InResponseTo="ID_2d10e285-42d4-4926-984e-fab8ea72d32a"
2146             IssueInstant="2004-01-30T21:59:38.409Z"
2147             MajorVersion="1" MinorVersion="1"
2148             ResponseID="ID_6e7e07bf-f02b-4535-ab9c-f19f7f31a3bf"
2149             xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol">
2150             <samlp:Status>
2151                 <samlp:StatusCode Value="samlp:Success"/>
2152             </samlp:Status>
2153             <saml:Assertion
2154                 AssertionID="..."
2155                 IssueInstant="2004-01-30T21:59:38.409Z"
2156                 Issuer="00268dd0-2f77-11d8-a14c-b8a03c50a862"
2157                 MajorVersion="1" MinorVersion="1"
2158                 xmlns:saml="urn:...:SAML:1.0:assertion">
2159                 <saml:Conditions
2160                     NotBefore="2004-01-30T21:59:38.409Z"
2161                     NotOnOrAfter="2004-01-31T02:59:38.409Z"/>
2162                 <saml:AuthorizationDecisionStatement
2163                     Decision="Deny"
2164                     Resource="urn:mydomain:HelloWorldService"
2165                     <saml:Subject>
2166                         <saml:NameIdentifier
2167                             Format="...:X509SubjectName"
2168                             NameQualifier="...">
2169                             CN=John Doe,OU=NCES,DC=DISA,DC=mil
2170                         </saml:NameIdentifier>
2171                     </saml:Subject>
2172                     <saml:Action>helloWorld</saml:Action>
2173                 </saml:AuthorizationDecisionStatement>
2174             </saml:Assertion>
2175         </samlp:Response>
2176     </soapenv:Body>
2177 </soapenv:Envelope>

```

A.5 XACML Policy Set

```

2180 <?xml version="1.0" encoding="UTF-8"?>

```

```

2181 <PolicySet PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-
2182 combining-algorithm:deny-overrides" PolicySetId="898b2a0f-6302-4b0a-
2183 818c-c9c018b30116" xmlns="urn:oasis:names:tc:xacml:1.0:policy">
2184   <Target>
2185     <Subjects>
2186       <AnySubject xsi:type="xsd:string"/>
2187     </Subjects>
2188     <Resources>
2189       <Resource>
2190         <ResourceMatch
2191           MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
2192             <AttributeValue
2193               DataType="http://www.w3.org/2001/XMLSchema#string">b462ce60-3adc-11d8-
2194               ba00-b8a03c50a862</AttributeValue>
2195             <ResourceAttributeDesignator
2196               AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
2197               DataType="http://www.w3.org/2001/XMLSchema#string" Issuer="00268dd0-
2198               2f77-11d8-a14c-b8a03c50a862" MustBePresent="false"/>
2199             </ResourceMatch>
2200           </Resource>
2201         </Resources>
2202         <Actions>
2203           <AnyAction xsi:type="xsd:string"/>
2204         </Actions>
2205       </Target>
2206       <Policy PolicyId="0dea2874-ad60-4097-baf0-6d0184e96257"
2207         RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
2208         algorithm:deny-overrides">
2209         <Target>
2210           <Subjects>
2211             <Subject>
2212               <SubjectMatch
2213                 MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
2214                   <AttributeValue
2215                     DataType="http://www.w3.org/2001/XMLSchema#anyURI">mydomain\john_doe</A
2216                     ttributeValue>
2217                   <SubjectAttributeDesignator
2218                     AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
2219                     DataType="http://www.w3.org/2001/XMLSchema#anyURI" Issuer="00268dd0-
2220                     2f77-11d8-a14c-b8a03c50a862" MustBePresent="false"
2221                     SubjectCategory="user"/>
2222                   </SubjectMatch>
2223                 </Subject>
2224             </Subjects>
2225             <Resources>
2226               <Resource>
2227                 <ResourceMatch
2228                   MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
2229                     <AttributeValue
2230                       DataType="http://www.w3.org/2001/XMLSchema#string">b462ce60-3adc-11d8-
2231                       ba00-b8a03c50a862</AttributeValue>
2232                     <ResourceAttributeDesignator
2233                       AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
2234                       DataType="http://www.w3.org/2001/XMLSchema#string" Issuer="00268dd0-
2235                       2f77-11d8-a14c-b8a03c50a862" MustBePresent="false"/>
2236                     </ResourceMatch>
2237                   </Resource>

```

```

2238         </Resources>
2239         <Actions>
2240             <Action>
2241                 <ActionMatch
2242 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
2243                     <AttributeValue
2244 DataType="http://www.w3.org/2001/XMLSchema#string">4e7336b8-bada-44cd-
2245 8052-4d811945ad36</AttributeValue>
2246                     <ActionAttributeDesignator
2247 AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
2248 DataType="http://www.w3.org/2001/XMLSchema#string" Issuer="00268dd0-
2249 2f77-11d8-a14c-b8a03c50a862" MustBePresent="false"/>
2250                     </ActionMatch>
2251                 </Action>
2252             </Actions>
2253         </Target>
2254     </Policy>
2255     <Policy PolicyId="50814dcc-9566-4cc6-8e5f-bf8070f48d45"
2256 RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
2257 algorithm:deny-overrides">
2258         <Target>
2259             <Subjects>
2260                 <Subject>
2261                     <SubjectMatch
2262 MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
2263                         <AttributeValue
2264 DataType="http://www.w3.org/2001/XMLSchema#anyURI">CN=John
2265 Doe ,OU=NCES ,DC=DISA ,DC=mil</AttributeValue>
2266                         <SubjectAttributeDesignator
2267 AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
2268 DataType="http://www.w3.org/2001/XMLSchema#anyURI" Issuer="00268dd0-
2269 2f77-11d8-a14c-b8a03c50a862" MustBePresent="false"
2270 SubjectCategory="user"/>
2271                     </SubjectMatch>
2272                 </Subject>
2273             </Subjects>
2274         </Resources>
2275         <Resource>
2276             <ResourceMatch
2277 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
2278                 <AttributeValue
2279 DataType="http://www.w3.org/2001/XMLSchema#string">b462ce60-3adc-11d8-
2280 ba00-b8a03c50a862</AttributeValue>
2281                 <ResourceAttributeDesignator
2282 AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
2283 DataType="http://www.w3.org/2001/XMLSchema#string" Issuer="00268dd0-
2284 2f77-11d8-a14c-b8a03c50a862" MustBePresent="false"/>
2285                 </ResourceMatch>
2286             </Resource>
2287         </Resources>
2288     </Actions>
2289     <Action>
2290         <ActionMatch
2291 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
2292             <AttributeValue
2293 DataType="http://www.w3.org/2001/XMLSchema#string">88cf4686-a73e-457b-
2294 b723-81de192876c4</AttributeValue>

```

```

2295                                     <ActionAttributeDesignator
2296 AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
2297 DataType="http://www.w3.org/2001/XMLSchema#string" Issuer="00268dd0-
2298 2f77-11d8-a14c-b8a03c50a862" MustBePresent="false"/>
2299                                     </ActionMatch>
2300                                 </Action>
2301                             </Actions>
2302                         </Target>
2303                     </Policy>
2304                 <Policy PolicyId="ab693163-002d-4419-a778-221e47981032"
2305 RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
2306 algorithm:deny-overrides">
2307                     <Target>
2308                         <Subjects>
2309                             <Subject>
2310                                 <SubjectMatch
2311 MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
2312                                     <AttributeValue
2313 DataType="http://www.w3.org/2001/XMLSchema#anyURI">CN=John
2314 Doe,OU=NCES,DC=DISA,DC=mil</AttributeValue>
2315                                 <SubjectAttributeDesignator
2316 AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
2317 DataType="http://www.w3.org/2001/XMLSchema#anyURI" Issuer="00268dd0-
2318 2f77-11d8-a14c-b8a03c50a862" MustBePresent="false"
2319 SubjectCategory="user"/>
2320                                 </SubjectMatch>
2321                             </Subject>
2322                         </Subjects>
2323                         <Resources>
2324                             <Resource>
2325                                 <ResourceMatch
2326 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
2327                                     <AttributeValue
2328 DataType="http://www.w3.org/2001/XMLSchema#string">b462ce60-3adc-11d8-
2329 ba00-b8a03c50a862</AttributeValue>
2330                                 <ResourceAttributeDesignator
2331 AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
2332 DataType="http://www.w3.org/2001/XMLSchema#string" Issuer="00268dd0-
2333 2f77-11d8-a14c-b8a03c50a862" MustBePresent="false"/>
2334                                 </ResourceMatch>
2335                             </Resource>
2336                         </Resources>
2337                     </Actions>
2338                     <Action>
2339                         <ActionMatch
2340 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
2341                                     <AttributeValue
2342 DataType="http://www.w3.org/2001/XMLSchema#string">b083caca-27c5-422a-
2343 a7fb-8a6c4ac860ef</AttributeValue>
2344                                 <ActionAttributeDesignator
2345 AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
2346 DataType="http://www.w3.org/2001/XMLSchema#string" Issuer="00268dd0-
2347 2f77-11d8-a14c-b8a03c50a862" MustBePresent="false"/>
2348                                 </ActionMatch>
2349                         </Action>
2350                     </Actions>
2351                 </Target>

```



```
2352         </Policy>
2353     </PolicySet>
2354
2355
```

2355 **B. REFERENCES**

- [ANISH] Anish Bhimani, *Web Services – Not So Fast*, in Information Security, October 2002 issue.
<http://www.infosecuritymag.com/2002/oct/webservices.html>
- [DOTNET] Microsoft .NET Web Services Enhancements (WSE) 1.0 Development Guide
- [JAXRPC] Java API for XML-Based RPC (JAX-RPC) Specification 1.0
<http://java.sun.com/xml/jaxrpc/docs.html>
- [LA-ARCH] Liberty Architecture Overview, Version 1.1, January 15, 2003.
http://www.projectliberty.org/specs/archive/v1_1/index.html
- [RBAC] NIST Role Based Access Control (RBAC) Standard, Draft 4/4/2003, available at: <http://csrc.nist.gov/rbac>
- [RFC2119] S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. IETF RFC 2119, March 1997. <http://www.ietf.org/rfc/rfc2119.txt>
- [RFC2828] RFC 2828, Internet Security Glossary
<http://www.ietf.org/rfc/rfc2828.txt>
- [SAML] Security Assertion Markup Language
<http://www.oasis-open.org/committees/security/#documents>
- [SIMWEB] Stephen Dill et al., Self-Similarity in the Web, ACM Transactions on Internet Technology, Vol. 2, No. 3, August 2002, pp.205-223
- [SOAP] Simple Object Access Protocol 1.1
<http://www.w3.org/TR/SOAP/>
- [WS-ADDR] Web Services Addressing (WS-Addressing) specification, <http://www-106.ibm.com/developerworks/webservices/library/ws-add/>
- [WS-GLOS] Web Service Glossary
<http://www.w3.org/TR/ws-gloss/>
- [WS-I] WS-Interoperability Initiative
<http://www.ws-i.org/>
- [WSS] Web Services Security: SOAP Message Security Spec 1.0 (Community Draft), <http://www.oasis-open.org/apps/org/workgroup/wss/>
- [WSS-SAML] Web Services Security: SAML Token Profile (Draft 08, 12/16/2003), <http://www.oasis-open.org/apps/org/workgroup/wss/>
- [WSS-SAML-I] Web Services Security: SAML Interop 1 Scenarios (Draft 4, 01/30/2004), <http://www.oasis-open.org/apps/org/workgroup/wss/>
- [WS-SC] Web Services Secure Conversation Language (WS-SecureConversation), version 1.0, Dec 18, 2002
<http://msdn.microsoft.com/ws/2002/12/ws-secure-conversation/>
- [WSSTP] WS-Security SAML Token Profile, Working Draft 06, Feb. 21, 2003
<http://www.oasis-open.org/committees/wss/documents/WSS-SAML-06.pdf>
- [WST] Web Services Trust Language (WS-Trust), version 1.0, Dec 18, 2002
<http://msdn.microsoft.com/ws/2002/12/ws-trust/>

- [XACML] XML Access Control Markup Language (XACML), Version 1.1,
[http://www.oasis-
open.org/committees/tc_home.php?wg_abbrev=xacml](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml)
- [XKMS] XML Key Management Specification
<http://www.w3.org/TR/xkms>
- [XMLDSIG] XML Signatures Syntax and Processing
<http://www.w3.org/TR/xmlsig-core/>
- [XMLENC] XML Encryption Syntax and Processing
<http://www.w3.org/TR/xmlenc-core/>

2356